SPECIFIKACE

ČLENSKÁ SEKCE VODNÍ ZÁCHRANNÉ SLUŽBY PRAHA 15

Verze 1.0

Zpracovali:
Peter Fačko
Jakub Levý
Vojtěch Švandelík

Supervizor:
RNDr. Martin Svoboda, Ph.D.

Konzultant:
Lukáš Kordík

24. května 2023

Obsah

Glosář

Ceník je šablona pro platby organizátorům *Událostí*. 4, 17, 18, 30, 31

Dítě je *Osoba* typu „člen dítě“. 4, 13, 19, 20

IS je zkratkou pro *Systém*. 4–6, 12–16, 18–24, 27, 28, 30, 32, 38–40

Kvalifikace je *Vlastnost*, která slouží jako certifikát znalosti či schopnosti *Osoby*. 4, 16, 18, 31

Oprávnění je *Vlastnost*, která opravňuje *Osobu* vykonávat určité funkce na *Událostech*. 4, 16, 18

Organizace je Vodní záchranná služba ČČK Praha 15, pobočný spolek. 4–12, 14–17, 22, 23, 32, 38

Osoba je fyzická osoba interagující s *Organizací*; typicky se jedná o člena *Organizace*, externího spolupracovníka nebo rodiče dítěte účastnícího se *Událostí*. 4, 12–21, 28–33, 37, 40

Povolení je množina činností, které je možné v *IS* vykonávat. 4, 15, 28, 32, 40

Pozice je funkce, kterou plní organizátor *Události*. 4, 14, 17, 20, 29, 31

Systém je informační systém *Organizace* specifikovaný touto specifikací. 4, 5, 30–32, 42

Transakce je záznam o peněžním závazku mezi *Organizací* a *Osobou*. 4, 16, 19, 21, 29, 30, 32, 33

Trénink je pravidelná *Událost*, která je organizovaná *Organizací*; *Trénink* má trenéra (*Osobu* s *Kvalifikací* „trenér“) a účastníky. 4, 13, 19, 20

Událost je událost, která je z pohledu *Organizace* jí organizovaná. 4, 13, 14, 16–20, 28–31, 33, 40, 42

Uživatel je *Osoba* s přiřazeným *Účtem*. 4, 12, 13, 15, 17, 20, 22, 24, 30–32, 40

Účet je entita určená pro interakci *Osob* s *IS*. 4, 12, 15, 18, 19, 28, 30, 32

Vlastnictví je *Vlastnost*, která představuje věc patřící určité *Osobě*. 4, 16

Vlastnost je charakteristika *Osoby*, která je pro *Organizaci* relevantní. 4, 16–18, 29, 30, 40

1 Úvod

Vodní záchranná služba ČČK Praha 15 (dále jen „*Organizace*“) pracuje především s dětmi a mládeží, pro které pořádá kroužky mladého vodního záchranáře na bazénu v Hostivaři, Holešovicích a v Třebešíně. Dále zajišťuje lezecký kroužek a pořádá mnoho jednorázových událostí pro své členy i pro veřejnost v nejširším slova smyslu. Významnou událostí, na které se podílí dospělí vodní záchranáři v rámci integrovaného záchranného systému (dále jen „IZS“), je služba na Orlické přehradě, která je prováděna během letních prázdnin. Mezi další činnosti prováděné *Organizací* patří komerční činnost jako např. zajišťování závodů na volné vodě. Nedílnou součástí fungování *Organizace* je i vzdělávání členů v oboru vodního záchranářství, kde se členové připravují na získání dalších rozšiřujících kvalifikací.

V současné době je rozsáhlá správa agend *Organizace* rozdělena na dvě části. Většina agend je spravována ručně pomocí volně dostupných nástrojů pro kolaboraci (Google Docs[1] apod.) a elektronické komunikace (zejména e-mail). Pro část agend v současné době existuje informační systém, ten ale neobsahuje všechny potřebné funkce a jejich přidání by bylo komplikované.

Rozsah agend, které musí být spravovány, se za více než 25letou existenci *Organizace* mnohonásobně zvýšil. Vzhledem k dnešní velikosti členské základny, která čítá okolo 150 členů, se správa *Organizace* ukazuje jako velmi časově náročná se sklony k vytváření chyb při správě libovolné agendy.

Cílem tohoto projektu je implementace nového informačního systému (dále jen „*Systém*“ nebo „*IS*“), který sníží čas členů *Organizace* potřebný k administraci a správě agend. *IS* bude eliminovat zbytečné chyby (kontrola odpovídajících kvalifikací při přidělení člena na událost apod.). Důležitým aspektem *IS* pro *Organizaci* je i možnost získávání konkrétních dat pro účely kontrol a výkazů o proběhlých událostech bez nutnosti ručního hledání ve fyzických dokumentech, případně v neudržitelném množství evidenčních tabulek.

IS bude vytvořen na míru *Organizaci*, tj. konkrétně pro Vodní záchrannou službu ČČK Praha 15, která je pobočným spolkem zastřešujícího spolku Vodní záchranná služba ČČK (dále jen „VZS“), která sdružuje vodní záchranáře v ČR. Při rozmýšlení záměru jsme diskutovali možnost vytvořit *IS* generický tak, aby byl využitelný bez větších úprav i v dalších pobočných spolcích, ale došli jsme k závěru, že jiné pobočné spolky VZS fungují velice odlišně, a tak po univerzálním *IS* není poptávka.

[1] Google Docs je cloudový balík nástrojů vyvinutý společností Google. Obsahuje textový editor, tabulkový procesor a prezentační software, které jsou přístupné přes webový prohlížeč nebo mobilní aplikaci. Zásadní funkcí Google Docs je, že umožňují spolupráci více uživatelů na stejném dokumentu v reálném čase. Odkaz na oficiální stránky: https://google.com/docs/about.

Jednotlivé aspekty vybraných agend, které *Organizace* spravuje, jsou blíže popsány v Sekcích 1.1–1.4.

1.1 Správa členské základny

Fyzické osoby se mohou stát členy *Organizace*. Členství je několika typů. U každého typu členství je potřeba evidovat mnoho údajů. Kromě těch zřejmých, jako jsou údaje osobní či kontaktní, je potřeba uchovávat i informace o různých oprávněních, např. řízení specifického dopravního prostředku. Dále je potřeba evidovat zapůjčené osobní vybavení a získané kvalifikace, které mohou být externí, nebo interní vydané v rámci *Organizace*.

Každý člen má základní přístup do *IS* a část údajů si může spravovat sám, zbylé údaje spravuje jiný uživatel s adekvátním oprávněním. Změna údajů může probíhat i tak, že uživatel navrhne změnu a jiný uživatel s dostatečným oprávněním mu změnu schválí a provede. Členové jsou sdružováni ve skupinách, tj. dospělí, děti, trenéři, velitelé aj. Člen může být členem i více skupin, např. dospělí a trenér. Členství je využito pro autorizaci ve specifických částech *IS*.

Kromě členství, které umožní osobě autorizovat se v rámci *IS*, existují i různá další oprávnění (např. povolení měnit osobní údaje členů) pro práci v *IS*.

1.2 Organizace kroužků pro děti

Nejzákladnější činností *Organizace* je práce s dětmi. *Organizace* pořádá řadu aktivit pro děti a mladistvé od 8 do 18 let. Tyto aktivity mohou být buďto pravidelné (kroužky vodní záchrany, kroužky lezení apod.), nebo nahodilé (vodácký výlet apod.). Pořádání aktivit se pojí s obsáhlou agendou, v rámci které se musí řešit: přihlašování na aktivity, placení poplatků, vedení docházky, plánování aktivit z pohledu vedoucích (kdo bude trénovat který trénink) atd. Zásadní je podpora výkazů proběhlých událostí pro účely výplat.

1.3 Činnost v rámci IZS

Organizace, stejně jako další pobočné spolky VZS, se zapojuje do činnosti IZS, a to především během letních měsíců, kdy působí na vodních plochách (Orlická přehrada) v režimu 24/7. Kromě toho má *Organizace* zřízen výjezdový tým, který může být povolán pro asistenci Hasičskému záchrannému sboru. Z těchto důvodu je potřeba evidovat služby během letních měsíců a také složení a materiální vybavení výjezdového týmu.

1.4 Účast na kurzech a komerčních událostech

Kromě organizace událostí pro děti jsou komerční události druhou největší aktivitou *Organizace*. Typickým příkladem komerční událost je vodní dohled – řada spolků provozující vodní sporty (veslařství, kanoistika apod.) pořádá pravidelné závody, na kterých potřebují zajistit odborný vodní dohled, který je *Organizace* schopna zajistit.

Pro komerční události je potřeba spravovat evidenci, účast jednotlivých členů, určování pozic na událostech, vykazování atd. Navíc je potřeba vytvářet z těchto událostí statistiky pro účely půjčování dalšího vybavení jednotlivým členům. Systém přihlašování musí umožňovat přihlášení i na části těchto událostí a zároveň musí hlídat, kdo s jakou kvalifikací se může přihlásit na tu kterou pozici na události.

2 Současný stav

Stávající řešení *Organizace* se skládá ze dvou hlavních komponent, které jsou navázány na agendy *Organizace*. Agenda spojená s prací s dětmi (tj. pravidelné tréninky a nepravidelné události pro děti) je z valné většiny řešena prostřednictvím systému *Členská sekce VZS Praha 15*. Zbytek agend je obhospodařován především soustavou textových a tabulkových dokumentů v cloudovém prostředí Google Docs.

Členská sekce VZS Praha 15 je systém vyvinutý na míru pro účely *Organizace*. Systém vyvinul, nadále spravuje a rozšiřuje konzultant této práce Lukáš Kordík. Funguje jako webová aplikace přístupná přes webovou adresu https://clen.vzs-praha15.cz.

Aplikace využívá technologie, které byly běžné v době jejího vývoje. Její hlavní část je vytvořena v jazyce PHP[1] verze 5.6[2], jehož podpora skončila v roce 2017. Back-endová část nevyužívá přímo žádný existující framework, ale na některé dílčí operace využívá části frameworku Nette[3]. Front-endová část webu je tvořena pomocí HTML[4], CSS[5] a JavaScript[6]. Komunikace mezi back-endovou a front-endovou částí probíhá tradičně, skrze HTTP[7] požadavky při načtení stránky. Většina operací je ale řešena pomocí technologie AJAX[8], v rámci které jsou odesílány HTTP požadavky na pozadí bez přenačtení stránky.

[1] PHP (Hypertext Preprocessor) je programovací jazyk určený pro vývoj webových stránek. Jedná se o běžně používaný programovací jazyk pro tvorbu dynamických webových aplikací a služeb. Odkaz na oficiální stránky: https://php.net.

[2] Migrace projektu postaveného na starém PHP 5.6 na novější verzi nemusí být triviální záležitostí kvůli změnám v PHP 7, které nedodržují kompatibilitu. Více info na stránkách: https://www.php.net/manual/en/migration70.php.

[3] Nette je populární framework pro vývoj webových aplikací v PHP. Je navržen tak, aby urychlil vývoj, zvýšil bezpečnost a zlepšil údržbu aplikací tím, že poskytuje sadu opakovaně použitelných komponent a nástrojů, které pomáhají vývojářům vytvářet udržovatelné aplikace. Odkaz na oficiální stránky: https://nette.org.

[4] HTML (Hypertext Markup Language) je značkovací jazyk používaný pro strukturování webových stránek. Odkaz na oficiální stránky: https://www.w3.org/html.

[5] CSS (Cascading Style Sheets) je jazyk používaný ke stylizování webových stránek vytvořených v HTML. Odkaz na oficiální stránky: https://www.w3.org/Style/CSS.

[6] JavaScript je vysokoúrovňový interpretovaný programovací jazyk, který se používá především na webových stránkách k přidání interaktivity a dynamického chování. Odkaz na oficiální stránky: https://www.ecma-international.org/ecma-262.

[7] HTTP (Hypertext Transfer Protocol) je protokol, který používají webové prohlížeče a webové servery k vyžádání a doručení webových stránek. Odkaz na oficiální stránky: https://www.ietf.org/rfc/rfc2616.txt.

[8] AJAX (Asynchronous JavaScript and XML) je technologie k vytváření dynamických a interaktivních webových aplikací. Umožňuje webovým stránkám aktualizovat obsah, aniž by bylo nutné obnovit celou stránku, což může zvýšit odezvu aplikací a zrychlit je. Více informací je dostupných na stránkách MDN (Mozilla Developer Network): https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX.

Systém obsahuje několik uživatelských rolí, které určují, do jakých částí systému má uživatel právo vstupovat. V případě, že pro více uživatelských účtů je nastavený stejný e-mail a heslo, tak si uživatel po přihlášení může vybrat, za jakou osobu chce se systémem interagovat.

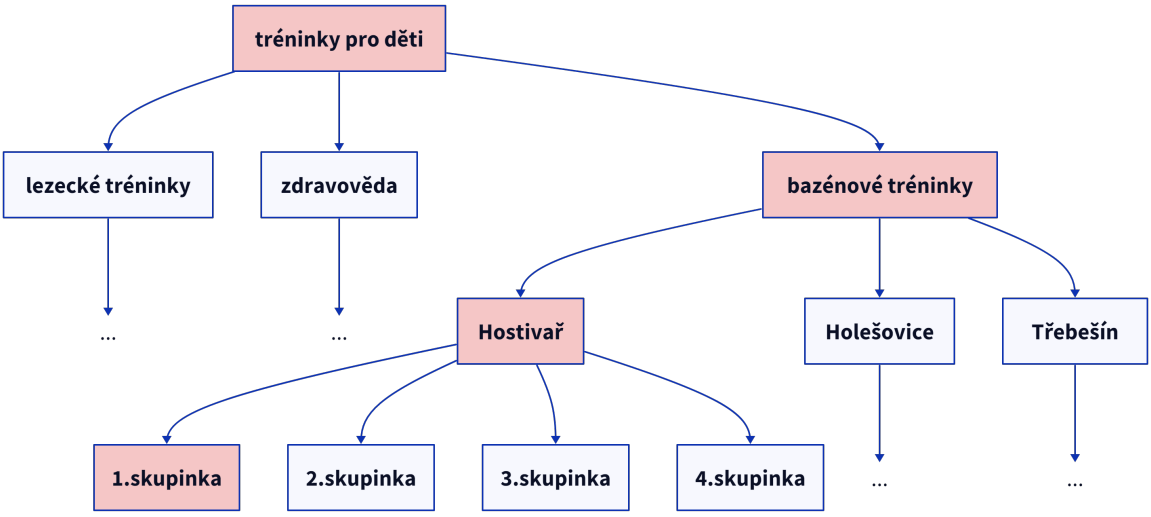
V případě, že uživatel nemá přiřazenou roli, jedná se o dítě (či rodiče dítěte), které se v systému může přihlašovat na tréninky a jednorázové události. Kromě toho zde vidí seznam plateb k uhrazení a možnost nastavení části osobních údajů svého profilu (e-mail, velikost oblečení, aj.).

Nejčastější rolí je trenérská role. Tato role umožňuje zobrazit výpis tréninků daného trenéra, odhlášení ze svého tréninku či přihlášení na trénink jiného trenéra v jeho nepřítomnosti. Dále trenéři mohou zapisovat docházku dětí na svých trénincích.

Další role jsou určeny pro správce systému a umožňují přidávání nových událostí, úpravu stávajících, správu uživatelů systému, úpravy plateb za události a přihlašování osob na události.

Účet v tomto systému mají jen děti účastnící se kroužků, trenéři a administrativní pracovníci *Organizace*. Osoby, které se věnují jen jiným typům aktivit, účet mít mohou, ale mohou si zde nastavovat jen část svých osobních údajů a pro tyto účely není jejich účet v systému potřeba.

Struktura dat v systému je stromová (týká se to především událostí) – události je možné zařazovat do stromu a tím přejímat některá nastavení či parametry rodičovských událostí. Příkladem může být následující struktura událostí: Bazénové tréninky – Bazénové tréninky Hostivař – Bazénové tréninky Hostivař 1. skupinka, kterou můžeme vidět na následujícím Obr. 2.1.



Obrázek 2.1: Struktura dat v systému

Google Docs Zbytek agend *Organizace* (především události pro dospělé členy a činnost pod IZS) je řešena spíše na nahodilé bázi prostřednictvím Google Docs. Centrálním místem je tabulka Účastí na události, ve které každý řádek odpovídá jedné události a do sloupců

se zapisují jednotlivé osoby, které se na události chtějí zapojit. Další komunikace ohledně konkrétních událostí pak probíhá na individuální ruční bázi pomocí e-mailových zpráv.

Kromě této tabulky existují i další, jako jsou tabulky Služby a Oprávnění. V tabulce Služby jsou evidováni dobrovolníci, kteří se zapojí do činnosti IZS na Orlické přehradě v letních měsících (do této tabulky zapisuje jen předsednictvo *Organizace* po vzájemné domluvě se zájemci). Tabulka Oprávnění slouží k evidenci osob, které mohou řídit vozidla dle typů, atd. K oběma těmto tabulkám má přístup veškerá dospělá členská základna *Organizace*.

Dále ještě existují tabulky a dokumenty, které obsahují údaje o členech, stav půjčování vybavení a další související údaje. K těmto tabulkám a dokumentům má přístup jen předsednictvo *Organizace*.

Tabulky a dokumenty jsou tříděny do složek a tyto složky jsou dále tříděny do sdílených disků, což je funkcionalita pro týmovou spolupráci v rámci produktu Google Drive[9]. Toto dělení částečně přispívá k přehlednosti těchto dat, ale jeho využívání je komplikované pro osoby, které nemají účet v rámci domény *Organizace* v sadě produktů Google Workspace[10].

Důvodů pro nový zastřešující systém je několik.

Zastaralost stávajícího řešení Zásadním problémem je, že stávající systém pro správu dětské agendy není rozvíjen dostatečně.

To znamená, že systém:

- používá technologie, které již nejsou bezpečné
- neobsahuje veškerou potřebnou funkcionalitu
- nemá responzivní grafiku, což komplikuje užívání webové aplikace na telefonu

Chybějící dokumentace Ke stávající webové aplikaci není žádná dokumentace a jelikož back-end nevyužívá komplexně nějaký existující zdokumentovaný framework, je velice obtížné najít další osoby, které by byly ochotné se rozvoji systému věnovat.

Nejednotnost systému Aby bylo možné *Organizaci* spravovat efektivně, bylo by potřeba mít všechna data a všechny procesy na jednom místě. To ve stávajícím řešení, které je tvořeno na jednu stranu vlastní webovou aplikací a na druhou stranu množinou tabulek, není možné.

[9] Google Drive je cloudová služba pro ukládání a synchronizaci souborů. Umožňuje uživatelům ukládat soubory v cloudu a přistupovat k nim z jakéhokoli zařízení s připojením k Internetu. Zásadní funkcí Google Drive je volitelné sdílení souborů včetně možnosti nastavení oprávnění pro konkrétní osobu. Odkaz na oficiální stránky: https://google.com/drive.

[10] Google Workspace je cloudový balík nástrojů, jeho součástí je řada nástrojů a služeb pro komunikaci a spolupráci. Příkladem služby v Google Workspace může být výše zmíněný Google Drive. Odkaz na oficiální stránky: https://workspace.google.com.

Nekonzistentnost dat Tím, že jsou využívány různé nepropojené tabulky, není téměř vůbec hlídaná konzistence dat. Například v případě odejití člena je potřeba ho odebrat z mnoha různých míst a mohou tak vzniknout velké nesrovnalosti v datové základně.

Nemožnost strojového zpracování *Organizace* potřebuje v určitých okamžicích datovou analýzu své činnosti. Příkladem takové analýzy může být zjištění, kolika událostí (a kolika hodin) se který člen zúčastnil v určitém roce. Získání těchto dat z tabulek, kde jsou využívány přezdívky a zkrácená jména, je obtížné a údaje se špatně exportují.

Při zohlednění všech těchto komplikací jsme usoudili, že rozvíjet stávající řešení není efektivní, a raději jsme se rozhodli pro vybudování řešení nového, které pokryje veškerou agendu a využije takové technologie, aby bylo možné systém dále rozvíjet dle potřeb *Organizace*.

3 Návrh řešení

3.1 Požadavky

Následující kapitola obsahuje základní funkční a nefunkční požadavky na *IS*. Požadavky byly sbírány od aktuálních členů *Organizace* a zachycují činnosti, které se v rámci *Organizace* provádějí.

Nutná funkcionalita systému je rozdělena do skupin požadavků, podle agend, které jsou danými požadavky řešeny. Jednotlivé skupiny požadavků mají v názvu uvedený prefix podle toho, jestli se jedná o funkční (FP) či nefunkční (NP) požadavky.

3.1.1 FP – Správa uživatelských účtů

Osoby budou s *IS* interagovat prostřednictvím svých uživatelských *Účtů*. Každá *Osoba* v *IS* má nejvýše jeden *Účet* a každý účet patří nějaké *Osobě*. *Uživatel* (*Osoba* s *Účtem*) může spravovat i jiné *Osoby*, než sebe (je potřeba mezi spravovanými *Osobami* vybírat). Spravování více *Osob* z jednoho *Účtu* má využití například pro rodiče s více dětmi.

Uživatelé mohou mít přidělené povolení v *IS* – např. „Předseda spolku“, „Správce členské základny“, „Správce tréninků“, atd.

3.1.2 FP – Evidence členské základny

IS bude obsahovat modul pro evidenci členské základny. *IS* bude evidovat všechny *Osoby*, které se zapojují do aktivit *Organizace* nebo potřebují mít přidělený *Účet*. *Osoby* budou moci být několika různých typů, přičemž typ se eviduje jen jako příznak vybíraný ze seznamu (každá *Osoba* má pouze jeden typ). O *Osobách* se dále evidují:

- osobní údaje
- kontaktní údaje
- dosažené kvalifikace
- zapůjčené vybavení
- oprávnění (povolení řídit daný dopravní prostředek, povolení vozit děti na události a další oprávnění dle aktuální potřeby)

Jednotlivé údaje k evidenci budou přidávány pouze pomocí zdrojového kódu – nebude možnost přidávat pole dynamicky pomocí *IS*.

Údaje bude moci spravovat *Uživatel* s povolením „Správce členské základny“. *Osoby* typu „člen dítě“ bude navíc moci upravovat *Uživatel* s povolením „Správce dětské členské základny“.

Každý *Uživatel* uvidí veškeré evidované údaje o své *Osobě* a bude moct změnit své kontaktní údaje.

3.1.3 FP – Pravidelné události pro děti

IS bude využitelný pro kompletní správu *Tréninků*. *Trénink* je pravidelná *Událost* pro *Osoby* typu „člen dítě“ (dále jen „*Děti*“). *Tréninky* budou několika typů a budou probíhat na několika lokalitách. Některé *Tréninky* mají více opakování v rámci týdne.

IS umožní například následující strukturu *Tréninků*: lezecké *Tréninky* probíhají třikrát týdně, vždy dva v jednom dni (jeden pro mladší žáky, jeden pro starší). *Dítě* si může vybrat, na kolik *Tréninků* týdně chce docházet a podle toho se počítá kapacita jednotlivých *Tréninků* a platby *Dítěte*; bazénové *Tréninky* probíhají na třech různých lokalitách, a to vždy dvakrát týdně, na dvě ze tří lokalit musejí *Děti* docházet na oba *Tréninky* týdně, na jednu lokalitu lze docházet jen jednou týdně. Struktura v průběhu školního roku zůstává zachovaná.

Děti se budou pomocí *IS* přihlašovat na *Tréninky* (vždy na celý školní rok), bude se evidovat jejich docházka a bude možné dělat dílčí úpravy termínů (např. zrušit *Trénink* v době prázdnin).

V případě, že se *Dítě* omluví z *Tréninku* ve lhůtě stanovené konstantou v nastaveni *IS*, *IS* mu umožní se přihlásit na náhradní termín v jiné lokalitě, ale daného typu.

V *IS* se evidují i závazky za *Tréninky*. Platební povinnost za celý školní rok je rozdělena na 3 části.

IS bude spravovat účast trenérů na jednotlivých termínech *Tréninku*. Na každém *Tréninku* musí být předepsaný počet trenérů s danou kvalifikací. Trenéři mají svůj rozvrh *Tréninků* po celý rok předem daný, ale *IS* umožňuje jejich výměnu v případě nepřítomností. Při přihlašování na konkrétní *Událost* trenérem se v *IS* ověřuje, jestli má kvalifikaci danou pro daný *Trénink* (typicky „trenér plavání“ či „trenér lezení“).

Tréninky, které probíhají vícekrát týdně, mohou mít ve výchozím stavu různé trenéry na různé dny. V případě neobsazenosti *tréninku* se danému výchozímu trenérovi pošle e-mailová notifikace.

Trenérům se evidují v *IS* odměny za absolvované *Události*.

3.1.4 FP – Nepravidelné události pro děti

IS bude umožňovat správu nepravidelných *Událostí* pro *Děti*. Pro každou *Událost* bude určeno, jaký je minimálního věk *Osob*, které se na tuto *Událost* mohou přihlašovat, a rovněž

bude určeno, jaké kvalifikace a oprávnění musejí splňovat dospělí členové pro přihlášení na tuto *Událost* jako organizátoři.

IS bude u účastníků i organizátorů *Událostí* evidovat jejich peněžní závazky a účast.

3.1.5 FP – Nepravidelné události pro dospělé členy

V *IS* budou spravovány události pro dospělé členy. Tyto *Události* budou mnoha typů. Typy se budou evidovat dynamicky v *IS*.

Bude se jednat především o kurzy, dobrovolné *Události*, komerční *Události*, atd.

U *Událostí* se bude evidovat základní popis (včetně místa, času, atd.). Dále budou u každé události evidovány *Pozice Osob*, které se této *Události* budou účastnit v určitém počtu. U každé *Pozice* mohou být definovány podmínky především pomocí kvalifikací, které omezí, které *Osoby* se mohou na *Událost* přihlásit (např. *Pozice* „řidič“ bude vyžadovat „řidičské oprávnění skupiny B“).

Osoby se mohou na *Událost* (*Pozici*) přihlásit jen na část doby (definováno pomocí půldnů), pokud to nastavení *Události* dovolí.

Dále se u *Události* bude evidovat docházka jednotlivých *Osob*, podle které se mj. bude vypočítávat odměna (stanovena dle *Pozice* účastníka na základě ceníku přiřazeného k *Události*). Do výpočtu odměny se rovněž zohledňují parametry z profilu *Osoby* (např. nejvyšší dosažená kvalifikace).

U některých *Událostí* (např. typu „kurz“) může být naopak evidován poplatek za účast.

3.1.6 NP – Technologie

IS vznikne jako webová aplikace s responzivním designem. Cílem je, aby bylo možné ovládat systém jak z prostřední webového prohlížeče na počítači (určeno především pro administrativu *Organizace*), tak i z mobilních telefonů (určeno především pro trenéry, kteří zapisují docházku přímo na události aj.).

3.1.7 NP – Dokumentace

IS musí mít přehledně strukturovanou technickou dokumentaci, která umožní externím subjektům *IS* nadále rozvíjet bez nutnosti kompletního prostudování hotových zdrojových kódů.

3.1.8 NP – Cenově dostupné hostování

IS musí být vytvořen tak, aby bylo možné ho hostovat u komerčních dodavatelů hostingových a cloudových služeb za přijatelné ceny pro neziskovou organizaci (např. cloudový poskytovatel Microsoft Azure).

3.1.9 NP – Přenos dat z původního systému

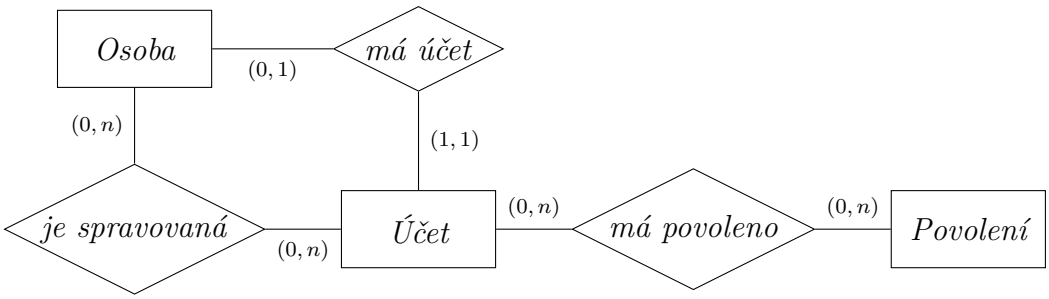
Pro zachování vhodné návaznosti na staré řešení musí dojít k přenosu alespoň části dat do nového *IS* z původní webové aplikace. Přenášet se budou uživatelské *Účty*, členská základna a návaznost mezi *Účty* a konkrétními *osobami*.

3.2 Entity a vztahy

V následující sekci jsou popsané entity a jejich vztahy vyskytující se v *IS*, doplněné o ER diagramy. Názvy entit a vztahů z pohledu *IS* jsou psané kurzívou (např. *účastní se*). Názvy entit jsou navíc psané s velkým počátečním písmenem (např. *Účet*).

Všechny entity a vztahy jsou dynamické, pokud není uvedeno jinak. Dynamický objekt je objekt, který může být vytvořený, změněný či smazaný v průběhu fungování *IS*. Všechny atributy mají maximální kardinalitu 1.

3.2.1 Uživatelé

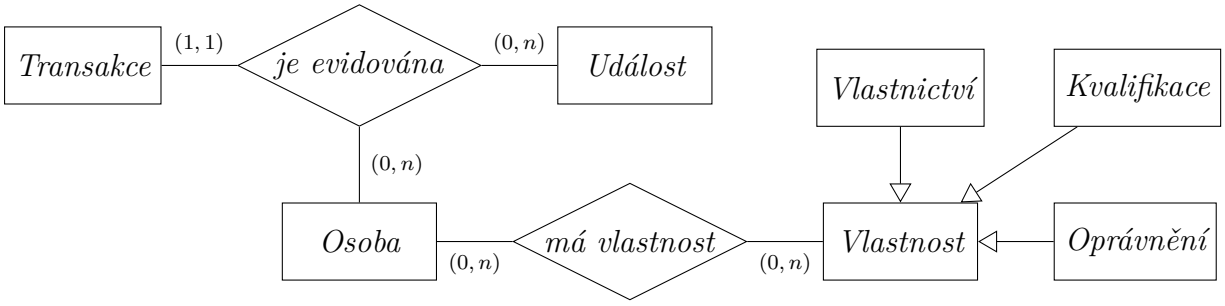


Účet je entita, skrze kterou přihlášená *Osoba* interaguje s *IS*. *Účet* je jednoznačně identifikován příslušnou *Osobou*. Interakce *Uživatele* s *IS* je bezpečná, což znamená, že *Účet* má přiřazeno heslo, které se používá k autentifikaci.

Typickými činnostmi prováděnými v *IS* jsou zobrazování a změna některých údajů v databázi. Tyto činnosti jsou seskupeny do konceptuálních množin, přičemž entita *Povolení* reprezentuje jednu takovou množinu. Pokud *má Uživatel povoleno* určité *Povolení*, je mu umožněno vykonávat odpovídající činnosti, přičemž přiřazení příslušného *Povolení* je jediný způsob, jak může *Uživatel* získat právo vykonávat nějakou činnost. Pro zjednodušení používání má každé *Povolení* přiřazený jedinečný název. Množina všech *Povolení* je statická.

Ačkoliv fyzické osoby relevantní pro *Organizaci* jsou reprezentovány entitou *Osoba*, přístup do systému je umožněn pouze prostřednictvím *Uživatele*. *Uživatel spravuje* vždy sám sebe, ale pro případ rodiče *spravujícího* své díte, je možné, aby *Uživatel spravoval* libovolný počet *Osob*. Pro rozvedené rodiče a jiné speciální případy je také možné, aby byla jedna *Osoba* spravována více *Uživateli*.

3.2.2 Osoby

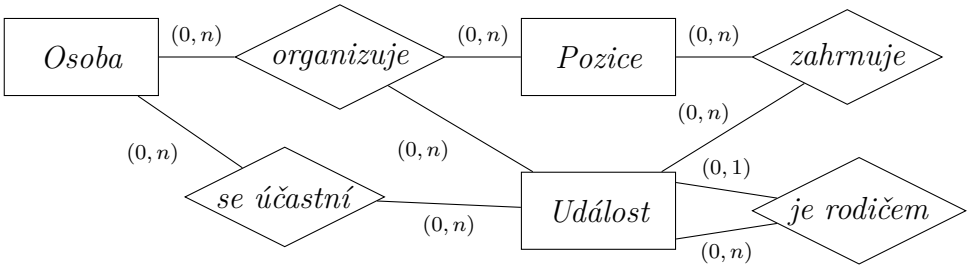


O *Osobách* uchovává *IS* pro identifikaci jejich jména, datum narození, rodné číslo a pohlaví. Kontaktní údaje uchovávané o *Osobách* jsou telefon a e-mailová adresa. *Osoby* jsou typicky členy *Organizace*, ale jelikož *IS* podporuje také správu jiných *Osob*, je nutné také ukládat jejich druh, který může být: „člen dítě“, „člen dospělý“, „člen čekatel“, „čestný člen“, „externí spolupracovník“ a „rodič“.

Osoba může *mít* různé *Vlastnosti*, což jsou libovolné charakteristiky relevantní pro *Organizaci*. *IS* potřebuje evidovat budoucí i minulá přiřazení *Vlastností Osobám*. Z toho důvodu je nutné pro vztah *mít vlastnost* evidovat čas přidělení a čas vypršení dané *Vlastnosti* s přesností na den. V období mezi těmito daty *má* daná *Osoba Vlastnost* platnou pro účely jejího *vyžadování*. Také je možné, aby přiřazení některých *Vlastností* nevypršela nikdy. *Vlastnosti* mohou být různých druhů: *Vlastnictví*, *Kvalifikace*, *Oprávnění*; přičemž všechny uchovávají své jméno a bližší určení kategorie. Také je u všech *Vlastností* uložený údaj o tom, zda daná *Vlastnost* expiruje. Tato informace je využívána pro automatické vyplnění údaje při vztahu *mít vlastnost*. Některé *Kvalifikace* mají mezi sebou uspořádání, které slouží při odměňování *Osob* – maximální kvalifikace určuje výši odměny. U *Kvalifikací* je tedy nutné evidovat číslo, které určuje jejich uspořádání.

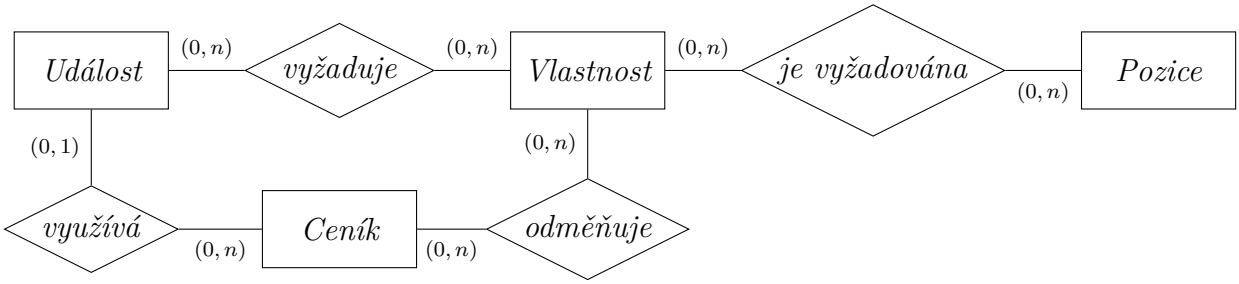
Osoba může být odměněna *Organizací* nebo jí může vzniknout vůči *Organizaci* dluh. Pro zaznamenání těchto finančních závazků slouží entita *Transakce*. Každá *Transakce* náleží právě jedné *Osobě* a obsahuje sumu, která značí odměnu dané *Osoby*. V případě záporné hodnoty se jedná o dluh *Osoby* vůči *Organizaci*. Dále *Transakce* obsahuje datum, od kterého je v platnosti, a text důvodu, ze kterého vznikla. Častým případem vzniku finanční odměny *Osobě* je účast na *Události*. Z toho důvodu může být *Transakce* spojena také s nejvýše jednou *Událostí*.

3.2.3 Události



Událost je událost, která je organizována z pohledu *Organizace*. To znamená, že *Událost* může být interní událost, organizovaná výhradně *Organizací*, nebo externí událost. Pokud samotná externí událost organizovaná *Organizací* není, organizován je výjezd *Osob* z *Organizace* na tuto událost. *Událost* obsahuje název, popis a termín začátku a konce konání. Po skončení *Události* je potřeba *Událost* uzavřít a toto uzavření schválit, proto *Událost* obsahuje také stav, který může být: „neuzavřená“, „uzavřená“ a „schválená“. Kvůli složité struktuře *Událostí* je možné přiřadit každé *Události* maximálně jednu *rodičovskou Událost*. Příkladem jsou instance pravidelného tréninku s jednou *rodičovskou Událostí*, která reprezentuje celý trénink.

Událost je potřeba *organizovat Osobami*. Každá *Událost zahrnuje* určité organizační role, které jsou reprezentovány entitami *Pozice*. *Osoba* pak *organizuje Událost* na určité *Pozici*, která má svůj název a popis. Každá *Událost zahrnuje* některé *Pozice* v určitém počtu, přičemž počet u jednotlivých *Pozic* určuje počet *Osob-organizátorů* vyžadovaných pro konání *Události*. *Organizovat Událost* je možné pouze na takových *Pozicích*, které jsou v ní *zahrnuté*. *Události* také umožňují *Osobám se* jich *účastnit*. Vztahy *organizuje* a *účastní se* mají význam pouze potenciální účasti, která musí být schválena jiným *Uživatelem*. Také je po skončení *Události* nutné vytvořit záznam o přítomnosti. Proto je atributem obou vztahů jejich stav: „čeká“, „schválený“, „náhradník“ nebo „přítomný“.



Účast na *Události* může být omezena více způsoby. Na *Osoby účastnící* se je kladeno omezení kapacitou a minimálním věkem účastníků, které jsou atributy *Události*. Od *Osob*, které *Událost organizují*, jsou *vyžadovány Událostí* určité *Vlastnosti* v různém počtu. Tyto požadavky mají účinek na celou *Událost*. Dále může mít *vyžadovány* určité *Vlastnosti* také každá *Pozice*.

Pro zjednodušení vyplácení organizátorů *Událostí* může každá *Událost* využívat maximálně jeden *Ceník*, který určuje platy. *Ceník* obsahuje základní plat, který je jeho atributem, a *odměňování* některých *Vlastností* určitou sumou.

3.3 Procesy

V této kapitole jsou popsány základní procesy, které budou v *IS* prováděny. Jednotlivé procesy jsou popsány textovou formou a jsou rozděleny do skupin dle agendy, které se týkají.

3.3.1 Členská základna

Zakládání nové osoby „Správce členské základny“ vyplní v *IS* údaje o nové *Osobě* na základě papírové přihlášky, kterou *Osobě* poskytl. Má možnost novou *Osobu* přidat ke stávajícímu *Účtu*, nebo jí založit nový.

Odebírání osoby „Správce členské základny“ v *IS* vybere *Osobu* ke smazání a ze systému ji odebere. Historie *Osoby* zůstane zachovaná, ale odeberou se osobní a kontaktní údaje o *Osobě*.

Přidání uživatele „Správce členské základny“ má možnost přidat ke stávajícímu profilu *Osoby* více než jeden uživatelský *Účet* (např. separátní *Účet* pro dva rodiče, či rodiče a dítě).

Úprava osoby „Správce členské základny“ může měnit osobní a kontaktní údaje *Osoby*, přidávat *Oprávnění* (např. „řídit vozidlo typu A“) a zadávat *Kvalifikace*. Zadávání *Oprávnění* probíhá výběrem ze seznamu předefinovaných *Oprávnění* v systému. *Kvalifikace* se zadávají pomocí vybrání typu *Kvalifikace* z číselníku a zadání platnosti kvalifikace. *Osoba* si sama může měnit jen své kontaktní údaje.

Půjčování vybavení „Správce členské základny“ do *IS* zadává zapůjčení konkrétního vybavení dané *Osobě* a uvádí, od kdy do kdy je vybavení zapůjčeno. Lze zadávat i časově neomezené zápůjčky.

Zobrazení oprávnění „Správce členské základny“ si může zobrazit v tabulce, jaké *Osoby* mají jaké *Oprávnění*. V případě shlukování *Oprávnění* do skupin (podle kategorie), je možné zobrazení *Oprávnění* pro celou skupinu (např. všechna „motorová vozidla“).

3.3.2 Události pro děti

Založení pravidelné události „Správce bazénových tréninků“ či „Správce lezeckých tréninků“ má možnost zadat údaje o nové pravidelné *Události*.

Přihlášení na pravidelnou událost Dítě, které je v systému již evidováno, si v *IS* zobrazí seznam dostupných *Událostí* a může se na konkrétní *Událost* přihlásit.

Dítě, které prozatím není v systému, si ve veřejné části *IS* vyplní přihlášku, kde si vybere, kam chce docházet, a zároveň si i vytvoří uživatelský *Účet*. Takto založená *Osoba* bude typu „člen dítě“.

V případě přihlášení na *Událost* s naplněnou kapacitou se *Dítě* zařadí mezi náhradníky.

„Správce dětské členské základny“ vidí seznam neschválených přihlášek a může je jednotlivě či hromadně schvalovat. Po schválení přijde dítěti e-mail s platebními informacemi.

Přihlášení na nepravidelnou událost *Dítě* si po přihlášení do *IS* zobrazí seznam nepravidelných *Událostí* a může se na dostupné *Události* přihlásit. Dostane e-mailem shrnutí základních informací vč. případných platebních údajů a kontaktu na hlavního organizátora.

V případě přihlášení na plnou *Událost* se *Dítě* zařadí mezi náhradníky.

Schvalování přihlášek na pravidelné i nepravidelné události „Správce tréninků“ si v detailu *Události* může zobrazit seznam přihlášených *Osob* a *Osob* zařazených mezi náhradníky a může přihlášky schvalovat a povyšovat náhradníky na přihlášené *Osoby*.

Export kontaktních údajů „Správce tréninků“ si může jednoduchým způsobem vygenerovat kontaktní údaje na *Děti* přihlášené na konkrétní *Trénink* či konkrétní aktivitu (lezení / bazény).

Rovněž lze exportovat osobní a kontaktní údaje u nepravidelných *Událostí*.

3.3.3 Pravidelné události pro děti (tréninky)

Úprava plateb za trénink „Správce tréninků“ si může zobrazit *Transakce* spojené s *Tréninky* a hromadně je upravit pro konkrétní skupinu *Tréninků* (například konstatovat, že platba za „3. čtvrtletí bazénových tréninků v Hostivaři“ bude vyšší o 100 Kč).

Přidání trenéra k tréninku „Správce bazénových tréninků“ či „Správce lezeckých tréninků“ má možnost k danému *Tréninku* (na celý rok či ke konkrétnímu termínu *Tréninky*) přiřadit trenéra. Trenér se také může zapsat sám na volné *Tréninky*.

Náhrada trenéra Trenér má možnost se z daného termínu *Tréninku* odhlásit do lhůty definované konstantou v *IS*. V takovém případě se ostatním trenérům e-mailem a v *IS* nabídne daný termín a mohou se na něj přihlásit. Změna se projeví v odměnách trenérům za *Tréninky*.

Zapisování docházky Po každém *Tréninku* zapíše trenér daného termínu *Události* docházku *Dětí* na *Událost*. V docházce vidí *Dětí* přiřazené k tomuto termínu. „Správce tréninků“ a určený pravidelný trenér *Tréninku* si mohou zobrazit docházku skupiny za celé pololetí, případně celý rok.

Upozornění na časté absence V případě, že se dítě několikrát po sobě nezúčastní *Tréninku*, dorazí e-mailová notifikace „Správci tréninků“. Počet *Tréninků* pro notifikaci je dán nastavením *IS*.

3.3.4 Události pro dospělé členy

Založení nové události pro dospělé členy – kurzy „Správce kurzů“ má právo založit *Událost* v kategorii „kurz“, kde vyplní základní údaje o *Události* vč. kapacity účastníků.

Založení nové události pro dospělé členy – komerční události „Správce komerčních událostí“ má právo založit *Událost* v kategorii „Komerční událost“, kde vyplní základní údaje o *Události*. Rovněž může zadávat *Pozice* na *Události* a definovat kritéria podle kterých se mohou osoby na *Událost* hlásit. Může zadat výši platu za každou *Pozici* na *Události*.

Přihlášení dospělého na událost Dospělý člen si může zobrazit všechny *Události*. U každé *Události* se může podívat na podrobnosti a vybrat si, na jakou *Pozici* se zapisuje.

Změna účastníků události Správce dané kategorie *Událostí* má možnost upravovat účastníky *Události* (přidávat či odebírat kohokoliv z *Osob*). Dané *Osobě* vždy přijde e-mailové upozornění.

Uzavření události Správce dané kategorie *Událostí* či *Osoba* v roli „velitel“ může po *Události* provést uzavření *Události*. Uzavření spočívá také ve vyplnění docházky. Po uzavření události ze strany velitele čeká *Událost* na uzavření (schválení) ze strany správce dané kategorie *Událostí* (může provést libovolný *Uživatel* s touto funkcí). Správce může jednotlivým účastníkům udělat změny v odměnách za *Událost*.

Při neuzavření *Události* do konstanty dané *IS* se veliteli *Události* a správci kategorie pošle e-mailová notifikace.

Přidání jednorázové odměny „Předseda“ může udělit libovolné *Osobě* kladnou *Transakci* (odměnu) či zápornou *Transakci* (poplatek) s popisem.

3.3.5 Transakce

Zobrazení transakcí Libovolná *Osoba* si může zobrazit seznam *Transakcí*, kde vidí, kolik peněz musí uhradit a kolik peněz naopak dostane.

Kontrola plateb *IS* pro každou *Transakci* vygeneruje QR kód. Automaticky páruje *Transakce* s platbami na bankovním účtě a po přijetí platby pošle dané osobě e-mailové potvrzení.

V případě platbě po splatnosti se rovněž pošle notifikace osobě.

Exportování platebních podkladů Účetní si může vyexportovat platební podklady pro všechny *Osoby*.

4 Implementace

V této kapitole se podíváme na náš konkrétní návrh implementace *IS*. Nejprve si ukážeme v Sekci 4.1 architekturu řešení a to včetně popisu a výběru technologií. Následuje Sekce 4.2 zabývající se datovým modelem. Poslední dvě Sekce 4.3, 4.4 se zabývají use-cases a návrhem uživatelského rozhraní.

4.1 Architektura

Podle požadavků a s přihlédnutím k implementační náročnosti, jsme pro *IS* zvolili centralizovaný model klient-server.

IS se bude skládat z back-endu a front-endu. Na serveru poběží back-end, který bude pro běžné *Uživatele IS* neviditelný. Běžná uživatelská práce s *IS* bude probíhat pouze přes UI[1], které nabídne front-end. Pro napojení *IS* na další systémy bude back-end disponovat API[2].

Při přístupu *Uživatele* do *IS* server vždy pošle aktuální verzi front-endu na uživatelův počítač, čímž eliminujeme problémy s instalací a neaktuálními verzemi SW[3].

Tento přístup k návrhu samozřejmě trpí zásadním problémem, kterým je centralizovaná komponenta v podobě serveru. Uvažovali jsme i o distribuovaném peer-to-peer modelu, který jsme zavrhli kvůli jeho nevhodnosti pro řešení problému, který je sám o sobě centralizovaný (správa, evidence apod.). Dalším nevýhodou distribuovaného peer-to-peer modelu pro nás je i jeho vyšší implementační náročnost oproti standardnímu klient-server řešení.

4.1.1 Výběr technologií

Výběru konkrétních technologií jsme věnovali zvláštní pozornost a péči zejména pro budoucí použitelnost a udržitelnost výsledného produktu.

Na omezený počet finančních prostředků *Organizace* je nutné myslet nejen při výběru programovacích jazyků, ale i u výběru databázového serveru, který bude ukládat data *IS*. Např. použití MSSQL[4] vyžaduje licenci při použití v produkčním prostředí, cena licence je mimo možnosti *Organizace*.

[1] UI neboli uživatelské rozhraní zahrnuje grafické uspořádání a vizuální prvky systému, s nimiž uživatelé interagují.

[2] API neboli Application Programming Interface. Jedná se o soubor protokolů, rutin a nástrojů pro vytváření aplikací, které umožňují různým SW systémům vzájemnou komunikaci.

[3] SW zkratka pro software. Software označuje programy, které běží na počítačích.

[4] MSSQL neboli Microsoft SQL Server je komerční systém pro správu relačních databází vyvinutý společností Microsoft. Odkaz na oficiální stránky: https://microsoft.com/en-us/sql-server.

Back-end

Po zvážení požadavků, zejména požadavku jednoduché, rychlé a levné nasaditelnosti, jsme nejprve uvažovali o programovacím jazyku PHP. Použití PHP je možné považovat za rozhodnutí pro výběr vyzrálé technologie, u níž máme nejvyšší šanci, že budoucí správci budou schopni celý *IS* administrovat. Za obrovskou výhodu považujeme existenci nepřeberného množství hostingových služeb pro PHP v samotném českém prostředí, kdy není vyžadována téměř žádná správa. Po objednávce služby stačí pouze nakopírovat zdrojový kód *IS*, vyhneme se tak správě vlastního serveru, což potřebám *Organizace* vyhovuje.

Nevýhod vyplývajících z výběru PHP ovšem není málo. Samotný jazyk vznikl v roce 1995 a za dobu jeho vývoje v něm vznikla řada nesystémových částí a nejednotností. Navíc se ustálily způsoby členění kódu pro programování webových aplikací, které jsou novější než jazyk PHP. Z toho důvodu je nutný výběr frameworku, který bychom použili pro tvorbu back-endu.

Uvažovali jsme nad použitím frameworku Nette, který je dílem českého programátora Davida Grudla. Většina projektového týmu ale s tímto frameworkem nemá zkušenosti a Web neoplývá vyčerpávající dokumentací pro nezasvěcené, ale spíše komerčními školeními, které jsou mimo možnosti členů projektového týmu.

Nakonec jsme polevili z požadavku objednávky hostingu přímo pro danou technologii a rozhodli jsme, že dobrým kompromisem bude provozování vlastního serveru. Vzhledem k cenovým možnostem *Organizace* se bude pravděpodobně jednat o server virtuální. Naším cílem bylo najít technologii, která je dostatečně vyzrálá a má šanci přežít do budoucna, zároveň disponuje kvalitní a obsáhlou dokumentací na Internetu.

Jako programovací jazyk jsme zvolili Python[5], protože v něm umí programovat všichni členové projektového týmu. Pro Python existují dva populární frameworky, které naplňují naše očekávání – jedná se o Django[6] a Flask[7].

Při rozhodování mezi frameworky Django a Flask jsme narazili na nedostatek zkušeností s oběma technologiemi pro erudované rozhodnutí. Po detailních průzkumu jsme se ale shodli, že použití Flask je vhodné spíše pro menší aplikace, protože samo o sobě nabízí jenom velmi málo funkcí – jedná se spíše o micro-framework. Naopak Django poskytuje ORM (objektově relační mapování), díky kterému se můžeme zcela vyhnout psaní SQL[8] dotazů. Veškerý kód tak můžeme psát pouze v Python a Django bude za běhu generovat SQL výrazy, což nám umožňuje v budoucnu změnit databázový systém za jiný, pokud

[5] Python je vysokoúrovňový interpretovaný programovací jazyk. Je navržen tak, aby se snadno četl a psal, má jednoduchou a intuitivní syntaxi, která klade důraz na čitelnost a udržovatelnost kódu. Odkaz na oficiální stránky: https://python.org.

[6] Django je webový framework s vyvíjený v Python. Řídí se architektonickým vzorem MVC (model-view-controller) a je navržen tak, aby vývojářům pomohl rychle a snadno vytvářet webové aplikace. Odkaz na oficiální stránky: https://djangoproject.com.

[7] Flask je odlehčený webový framework vytvořený v Python. Je navržen s jednoduchým a minimalistickým API, které vývojářům umožňuje snadno vytvářet webové aplikace. Odkaz na oficiální stránky: https://flask.palletsprojects.com.

[8] SQL neboli Structured Query Language je jazyk určený pro správu relačních databází a manipulaci s nimi.

ho bude Django podporovat. Kromě vyhnutí se psaní SQL dotazů považujeme za výhodu i ochranu proti útokům typu SQL injection, které si Django samo ošetřuje.

Front-end

U front-endu bylo nejprve nutné rozhodnout koncepčně, jak moc spjatý s back-endem ho chceme mít. Na výběr máme ze dvou možností, buďto můžeme mít back-end zcela oddělen od front-endu, nebo je možné celý systém navrhnout jako monolit, tj. spojit back-end s front-endem.

Moderní způsob návrhu webových aplikací preferovaný mnoha webovými inženýry v současné době je oddělení front-endu od back-endu. Front-end tak může být tvořen grafickými designery a front-endovými programátory nezávisle na back-endových inženýrech. Běžně se pro návrh UI používají navíc ke standardním technologiím HTML, CSS, JavaScript i frameworky – obvykle v dnešní době React[9], nebo Vue.js[10]. V tomto případě je ale vhodné pro vývoj back-endu použít micro-frameworky, které nenabízí mnoho funkcí jako šablonovací systémy, protože je stejně nevyužijeme.

Navrhnout webovou aplikaci jako monolit patří mezi běžné způsoby vývoje, které jsou obvykle podpořeny šablonovacím nástrojem frameworku pro back-end, v našem případě Django disponuje vlastním šablonovacím nástrojem a dostatečně již ulehčuje práci s front-endem. Naše volba padla na monolitické řešení a tvorbu front-endu za použití standardních technologií HTML, CSS a JavaScript s pomocí Django.

Django ale neobsahuje vestavěné komponenty pro UI a paletu vhodných barev pro rychlou tvorbu komponent. Namísto tvorby vlastních komponent, u kterých bychom museli testovat UX[11] uživatelů, jsme se rozhodli pro použití existující šablony založené na frameworku Bootstrap[12].

Považujeme to za vhodnější, protože žádný s členů nemá dostatečnou erudici v grafickém designu pro vlastní tvorbu, která ve výsledku může způsobit špatné UX uživatelů při používání systému. Pro kontrolu kvality při tvorbě UI máme v plánu provádět uživatelské testování *IS* a dle zpětné vazby ho případně upravovat.

Při průzkumu existujících šablon jsme narazili na nedostatečné zkušenosti členů projektového týmu s vývojem front-endu. Z nepřeberného množství existujících šablon vyskytujících se na Internetu jsme nakonec vybrali šablonu AdminLTE[13].

[9] React je populární knihovna JavaScript, která se používá k vytváření UI pro webové aplikace. Odkaz na oficiální stránky: https://react.dev.

[10] Vue.js je stejně jako React knihovna pro vytváření UI. Jedná se o jednu z nejznámějších alternativ React. Odkaz na oficiální stránky: https://vuejs.org.

[11] UX je zkratka pro User Experience. Odkazuje na celkový zážitek, který má *Uživatel* při interakci s produktem, jako jsou webové stránky, mobilní aplikace apod.

[12] Bootstrap je knihovna pro vývoj UI webových stránek navržená tak, aby zjednodušila proces vytváření responzivních a mobilních aplikací. Její součástí je soubor předpřipravených komponent, jako jsou navigační lišty, formuláře, tlačítka, modální okna a další, které lze snadno přizpůsobit a uspořádat tak, aby vytvářely komplexní a responzivní rozvržení. Odkaz na oficiální stránky: https://getbootstrap.com.

[13] Odkaz na oficiální stránky: https://adminlte.io.

AdminLTE je šablona založená na Bootstrap 4, která zahrnuje obrovské množství široce testovaných komponent včetně palet barev. Zásadní výhodou AdminLTE pro nás je její rozšířenost a množství volně dostupné dokumentace na Internetu, díky které je práce s ní vhodná i pro vývojáře bez zkušeností s vývojem front-endu.

Databázový systém (DB systém)

Výběr DB systému pro nás znamenal nejprve rozhodnout, se zda použijeme klasické relační SQL databáze, nebo využijeme NoSQL[14] databáze. Rozhodli jsme se pro použití SQL databáze, z důvodu nedostatku zkušeností s NoSQL databázemi. Navíc samotné SQL nemusíme řešit díky ORM frameworku Django, které oficiálně podporuje SQL relační databáze PostgreSQL[15], MariaDB[16], MySQL[17], Oracle DB[18] a SQLite[19].

Výběr samotného DB systému není příliš podstatný díky mezivrstvě ORM. Z podporovaných databázových systémů určitě můžeme vyřadit Oracle DB kvůli licenčním problémům, SQLite, které je nevhodné pro rozsáhlé systémy, kde může docházet k paralelním přístupům.

Favoritem pro nás je PostgreSQL, který je možné použít pod svobodnou licencí. Navíc se jeho vývoj stále aktivně posouvá kupředu. PostgreSQL je pro nás stabilní DB systém s dostatečnou implementací podpory jazyka SQL a přijatelnou paměťovou náročností celého DB systému.

4.1.2 Technologie

Nyní se zaměříme na konkrétní vybrané technologie, o kterých si povíme více.

Python[20]

Python je vysokoúrovňový interpretovaný programovací jazyk, který byl poprvé vydán v roce 1991 nizozemským programátorem Guido van Rossumem. Python je známý pro svou

[14] NoSQL je druh databází, které jsou vhodné pro zpracování velkých objemů nestrukturovaných nebo částečně strukturovaných dat. Na rozdíl od tradičních relačních databází, které používají strukturované tabulky s předem definovanými schématy, databáze NoSQL obvykle používají flexibilní schéma, která se mohou přizpůsobit různým datovým formátům a strukturám.

[15] PostgreSQL, známý také jako Postgres, je relační DB systém, který klade velký důraz na spolehlivost a shodu s SQL. Odkaz na oficiální stránky: https://postgresql.org.

[16] MariaDB je relační DB systém, který je navržen jako náhrada DB systému MySQL. MariaDB vytvořili původní vývojáři MySQL poté, co ji koupila společnost Oracle. Odkaz na oficiální stránky: https://mariadb.org.

[17] MySQL je stále ještě často používaným relačním DB systémem, ačkoliv se část komunity přesunula k MariaDB poté, co společnost Oracle koupila MySQL. Odkaz na oficiální stránky: https://mysql.com.

[18] Oracle DB je komerční relační DB systém vyvinutý společností Oracle. Odkaz na oficiální stránky: https://oracle.com/database.

[19] SQLite je relační DB systém, který se téměř výhradně používá v malých aplikacích a vestavěných systémech. Na rozdíl od běžných relačních DB systémů, je SQLite bezserverový, samostatný databázový stroj, který je určen k zabudování přímo do aplikací. Odkaz na oficiální stránky: https://sqlite.org.

[20] Informace obsažené v této sekci pochází z oficiálních stránek Python, odkaz: https://python.org.

jednoduchou syntaxi, srozumitelnost, díky čemuž je oblíbeným jazykem pro začátečníky i zkušené programátory.

Python se používá v řadě různých aplikací, včetně vývoje webových stránek, vědeckých výpočtů, analýzy dat aj. V současné době se Python může chlubit velkou komunitu vývojářů a uživatelů, díky čemuž je k dispozici mnoho knihoven a nástrojů třetích stran, které rozšiřují jeho možnosti.

Na Obr. 4.1 můžeme vidět logo Python.

Obrázek 4.1: Logo Python

Django[21]

Django populární open-source webový framework vydaný pod svobodnou licencí BSD 3-Clause[22] postavený na programovacím jazyku Python. Poprvé byl uvolněn v roce 2005 s cílem zjednodušit proces vytváření webových aplikací.

Jádrem Django je architektonický vzor MVC (model-view-controller), který pomáhá vytvářet udržovatelné webové aplikace. Django však svou implementaci tohoto vzoru označuje jako „model-view-template“ (MVT), kde vrstva šablony (angl. template) odděluje prezentační logiku od zbytku aplikace.

Jednou z klíčových předností Django je důraz na rychlý vývoj a jednoduchost kódu. Dodržováním zásady DRY („Do Not Repeat Yourself“) podporuje Django psaní opakovatelně použitelného kódu, což vede ke zvýšení produktivity a snadnější údržbě aplikací v budoucnu.

Django následuje filozofii Python a stejně jako jazyk samotný obsahuje bohatou sadu funkcí a vlastností již v základní instalaci. Jako příklad můžeme uvést ORM (Object-Relational Mapping) pro práci s databázemi. ORM poskytuje abstrakční vrstvu, která vývojářům umožňuje komunikovat s databází pomocí Python, aniž by bylo nutné psát vlastní dotazy SQL. Aktuálně Django oficiálně podporuje několik DB systémů, konkrétně: PostgreSQL, MariaDB, MySQL, Oracle, SQLite.

U Django hraje významnou roli také jeho silný ekosystém. Má aktivní komunitu, která přispívá k jeho vývoji a udržuje mnoho rozšiřujících balíčků, díky kterým je možné např. použít ORM Django k připojení se i na MSSQL databázi.

[21] Informace obsažené v této sekci pochází z oficiálních stránek Django, odkaz: https://djangoproject.com.

[22] BSD 3-Clause je svobodnou licencí pro open-source SW. Jedná se o jednu z nejrozšířenějších licencí používaných pro open-source SW. Znění licence viz https://opensource.org/license/bsd-3-clause.

AdminLTE[23]

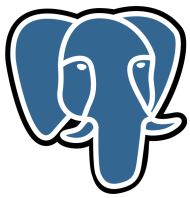
AdminLTE je open-source šablona dostupná pod licencí MIT[24] postavená na frameworku Bootstrap. Poskytuje řadu komponent uživatelského rozhraní, jako jsou grafy, tabulky, formuláře apod., které vývojářům pomáhají snadno a rychle vytvářet moderně vypadající responzivní webové aplikace. Nespornou výhodou je vysoká přizpůsobitelnost šablony, což vývojářům umožňuje upravit vzhled tak, aby odpovídal jejich specifické aplikaci. AdminLTE je mezi vývojáři oblíbená šablona a byla použita v řadě projektů. Jako příklad můžeme uvést ReCodEx, což je na MFF UK dobře známý systém pro analýzu a automatické vyhodnocování programátorských úloh.

PostgreSQL[25]

PostgreSQL je open-source databázový systém vydaný pod svobodnou licencí, který klade důraz na rozšiřitelnost a shodu s SQL. Poprvé byl vydán v roce 1989 a nyní je jedním z nejrozšířenějších DB systémů používaných v současnosti.

PostgreSQL je vysoce škálovatelný a podporuje také replikační a clusterové použití, je proto vhodný pro použití v kritických aplikacích. Zásadní odlišností oproti některým konkurenčním DB systémům je silný důraz na dodržování standardů. Je plně kompatibilní se standardy SQL, což usnadňuje práci při migraci dat při použití různých DB systémů, které dodržují standard SQL.

Na Obr. 4.2 můžeme vidět logo PostgreSQL.



Obrázek 4.2: Logo PostgreSQL

4.1.3 Rozdělení na Django aplikace

Django, které jsme si vybrali jakožto implementační framework, podporuje dělení projektu na další menší části, které nazývá aplikacemi. Tradiční dělení na back-end a front-end pro nás není ideální kvůli monolitickému návrhu systému. Rozhodli jsme se, že podle specifikace je možné *IS* rozdělit na tři aplikace: Users, Members a Events.

[23] Informace obsažené v této sekci pochází z oficiálních stránek AdminLTE, odkaz: https://adminlte.io.

[24] MIT licence je svobodnou licencí pro open-source software. Původně byla vyvinuta a používána MIT (Messachusettským technologickým institutem), po kterém je i pojmenována. Znění licence viz https://opensource.org/license/mit.

[25] Informace obsažené v této sekci pochází z oficiálních stránek PostgreSQL, odkaz: https://postgresql.org.

Users obsahuje logiku pro přihlašování, registraci *Osob* a ověřování *Povolení*.

Members zahrnuje správu členské základny, včetně definování a upravování skupin.

Events zajišťuje vše ohledně *Událostí*: definování pravidelných i jednorázových *Událostí*, přihlašování účastníků na *Události* apod.

Pro zajištění kompletní funkcionality *IS* aplikace mezi sebou komunikují. Součástí každé aplikace je část back-endu a front-endu nezbytná pro její fungování.

4.2 Datový model

V této sekci implementujeme entity a vztahy navržené v Sekci 3.2. Django nabízí ORM, takže jejich implementací budou Python třídy dědící `models.Model`, přičemž každému modelu přísluší jedna tabulka v databázi.

4.2.1 Entity

Django poskytuje základní implementaci uživatele: `models.User`. Ta poskytuje autentifikaci a autorizaci (*Povolení*), což jsou důležité části naší entity *Účet*. Bohužel má určité vlastnosti, které našemu návrhu nevyhovují. Hlavní z těchto vlastností je používání username jako hlavního identifikátoru, zatímco *IS* vyžaduje jako identifikátor *Osobu*. Z toho důvodu implementujeme vlastní uživatelský model jako třídu dědící `AbstractUser`. `AbstractUser` v sobě zahrnuje autentifikaci a autorizaci, ale umožňuje přepsat sloupce.

Entita *Osoba* vyžaduje jako atribut jméno. Jméno osob je možné rozdělit do více sloupců, což umožňuje jednodušší operace na jednotlivých částech jména, například příjmení. Druhou možností je uložit jméno jen v jednom sloupci. Jelikož budeme osoby vypisovat a je vhodné mít výpisy řazené abecedně podle příjmení, budeme jméno ukládat jako dva oddělené údaje a tedy ve dvou sloupcích – křestní jméno a příjmení.

Některé atributy mají pouze několik platných hodnot. V těchto případech použijeme Django mechanismus `TextChoices` spolu s parametrem `choices` v `CharField`. Databáze tak bude obsahovat pouze obyčejné pole znaků, ale vrstva Django bude na sloupci provádět validaci na základě našich zadaných hodnot. Navíc `TextChoices` nabízí i možnost specifikovat podobu hodnoty sloužící k prezentaci.

Ve sloupcích typu `CharField` je nutné specifikovat jejich maximální délku. Náš *IS* používá PostgreSQL, tudíž máme možnost spoléhat se na implementační detail, že PostgreSQL má neomezenou délku `VARCHAR`, což je databázový typ používaný pro `CharField`, ale rozhodli jsme se délku i přesto specifikovat. V případě potřeby neomezené délky textu použijeme `TextField`, takže relativně malá maximální délka pro `CharField` je dostačující. Nejvyšší přenositelnost z hlediska databáze dosáhneme použitím hodnoty 255, jelikož

je to zaručeně fungující hodnota pro všechny databáze podporované Django.

Zbytek atributů specifikovaných u entit a vztahů má přímočarou implementaci: data a časové údaje využívají `DateField` a `DateTimeField`, logické atributy jsou implementovány pomocí `BooleanField` a celá čísla nabízí `IntegerField` a jeho varianty se specializovanější validací.

```
class Feature(models.Model):
    class Type(models.TextChoices):
        QUALIFICATION = "K", _("kvalifikace")
        POSSESSION = "V", _("vlastnictvi")
        PERMIT = "O", _("opravneni")

    feature_type = models.CharField(max_length=1, choices=Type.choices)
    category = models.CharField(max_length=20)
    name = models.CharField(max_length=50, unique=True)
    expires = models.BooleanField(default=True)
    tier = models.PositiveSmallIntegerField(default=0)
```

Obrázek 4.3: Příklad implementace entity *Vlastnost*

4.2.2 Vztahy

Binární vztah kardinality maximálně jeden k libovolnému počtu (one-to-many), například *Transakce je evidována*, implementujeme pomocí `ForeignKey` na entitě, která se účastní vztahu maximálně jednou. V případě, že se daná entita vztahu účastnit nemusí, například *rodič Událost*, nastavíme pro daný sloupec `null=True`.

Možnost sloupce nabývat hodnotu `null` je výchozím nastavením vypnutá, což nám vyhovuje prakticky ve všech modelech. Výjimkou je kromě některých one-to-many vztahů také entita *Událost*. Z důvodu sémantiky *rodičovství Událostí* je nutné povolit pro téměř všechny sloupce `null=True`. Výjimkou jsou textové sloupce, u kterých je zvykem používat místo `null` prázdný řetězec.

Binární vztah s neomezenou kardinalitou na obou stranách (many-to-many) je možné implementovat pomocí `ManyToManyField`. V případě potřeby atributů na tomto vztahu Django nabízí možnost implementovat vlastní join model s dvěma `ForeignKey`, pomocí kterého je daný vztah realizován. V tomto modelu je navíc nutné specifikovat `unique_together` na `ForeignKey`, které model obsahuje, jelikož o daném vztahu potřebujeme informaci pouze jednou. Některé many-to-many vztahy specifikují minimální kardinalitu 1. Toto omezení na úrovni databáze neimplementujeme, jelikož pro něj existují pouze nedokonalá řešení.

Náš návrh obsahuje jeden trojitý vztah. Jedná se o *Osoby organizující Událost* na určité *Pozici*. Jeho sémantika je taková, že *Osoba* může *organizovat Událost* pouze na takové *Pozici*, která je *zahrnuta* v dané *Události*. Z toho důvodu můžeme implementovat daný vztah jako binární many-to-many vztah mezi *Osobou* a join modelem pro vztah *zahrnuje*.

Pro binární vztah s kardinalitou omezenou na 1 na obou stranách (one-to-one), který

se vyskytuje mezi *Účtem* a *Osobou*, použijeme OneToOneField. Protože každý *Účet* má *Osobu* ale ne všechny *Osoby* mají *Účet*, zadáme tento sloupec v modelu pro entitu *Účet*.

Tři entity dědí entitu *Vlastnost*, přičemž pouze jedna z těchto entit má navíc jeden atribut. Z tohoto důvodu implementujeme všechny tyto entity pouze jedním modelem, který je jejich sjednocením. Toto sjednocení je implementačně snazší než samostatné implementace čtyř entit. Pro rozlišení jejich typů použijeme dodatečný sloupec `CharField` s mechanizmem `TextChoices`.

Při použití `ForeignKey` máme možnost nastavit chování při odstranění odkazovaného objektu pomocí mechanismu `on_delete`. Jednou možností je všude použít `RESTRICT`, které má za úkol zabránit odstranění objektu, dokud na něj někdo odkazuje. Na opačném konci spektra je použití `CASCADE`, které při odstranění odkazovaného objektu odstraní také všechny odkazující objekty. V naší implementaci volíme možnost, že odstranění v databázi je vždy korektní, a proto ve většině případů používáme `CASCADE`. To má smysl zejména u spojovacích modelů vztahů, kde bez existence entit ve vztahu má záznam o vztahu malý význam. Výjimkou použití `CASCADE` je odkaz na *Událost* v případě *Transakce* a odkaz na *Ceník* v případě *Události*, kde se jedná o doplňující informaci, bez které je entita stále platná. Na těchto místech použijeme `SET_NULL`, který nastaví sloupec na `null`.

```
class FeatureAssignment(models.Model):
    person = models.ForeignKey(Person, on_delete=models.CASCADE)
    feature = models.ForeignKey(Feature, on_delete=models.CASCADE)
    date_assigned = models.DateField()
    date_expire = models.DateField()

    class Meta:
        unique_together = ["person", "feature"]
```

Obrázek 4.4: Příklad implementace vztahu *má vlastnost*

4.3 Use-cases

V této kapitole na několika reprezentativních příkladech ukážeme možný průchod webovou aplikací *Systému* ve snaze docílit určitých kroků. Příklady jsou vybírány tak, aby bylo možné si vytvořit ucelenou představu o běžné práci *Uživatelů* v *IS* a nejedná se o kompletní seznam všech možných činností.

Zadání nové komerční události

1. „Správce komerčních událostí“ se přihlásí do *IS*.
2. V menu klikne na záložku **Události**.
3. V horní části klikne na **Přidat novou událost**.
4. Do otevřeného editačního formuláře vyplní název, popis *Události* a data jejího trvání.

5. Ze seznamu vybere typ *Události* (**Komerční událost**) a z dalšího seznamu vybere *Ceník* pro danou *Událost*.

6. V sekci **Organizátoři** klikne na tlačítko **Přidat pozici** a do otevřeného modálního okna zadá údaje o *Pozici* „velitel“ (název *Pozice*, minimální *Kvalifikace*).

7. V sekci **Organizátoři** klikne na tlačítko **Přidat volnou pozici** a do otevřeného modálního okna zadá údaje o *Pozici* „řidič lodi“ (počet *Osob* na *Pozici*, minimální *Kvalifikace*).

8. Pomocí tlačítka **Uložit** přidá novou *Událost* do *Systému*.

Přihlášení se na komerční událost

1. *Uživatel* se přihlásí do *Systému*.

2. V menu klikne na záložku **Událost**.

3. V seznamu nabízených *Událostí* si najde *Událost*, o kterou má zájem, a pomocí tlačítka **Detail** si otevře stránku s informacemi o *Události*.

4. Pomocí tlačítka **Přihlásit na volnou pozici** se na *Událost* přihlásí.

Uzavření události

1. *Uživatel* příslušný *Osobě*, která vystupovala na události na *Pozici* „velitel“ se přihlásí do *Systému*.

2. V menu klikne na záložku **Událost**.

3. V horní části stránky si v seznamu neuzavřených *Událostí Osoby* pomocí tlačítka **Uzavřít událost** zobrazí konkrétní neuzavřenou *Událost*.

4. Vyplní popis *Události* a zadá docházku jednotlivých *Osob* na *Událost*.

5. Pomocí tlačítka **Uzavřít** uzavření *Události* potvrdí. *Systém* pošle notifikaci všem „Správcům komerčních událostí“.

6. „Správce komerčních událostí“ se přihlásí do *Systému*.

7. Otevře si detaily dané *Události*.

8. Přečte si údaje vyplněné „velitelem“, volitelně v seznamu mezd upraví odměny jednotlivých *Osob* na *Události*.

9. Pomocí tlačítka **Uzavřít** uzavření *Události* potvrdí a zmizí ze seznamu neuzavřených *Událostí*.

Úprava vlastních údajů

1. *Uživatel* bez speciálních *Povolení* se přihlásí do *Systému*.
2. V menu klikne na záložku **Členská základna**.
3. Zobrazí se mu údaje o *Osobě* přiřazené ke svému uživatelskému *Účtu*.
4. Pomocí tlačítka **Upravit** si zobrazí editační formulář, kde může editovat své kontaktní údaje.
5. Pomocí tlačítka **Uložit změny** se nové údaje uloží.

Export transakcí

1. „Účetní“ se přihlásí do *Systému*.
2. V menu klikne na záložku **Transakce**.
3. Zobrazí se mu seznam všech *Transakcí* všech *Osob*.
4. Pomocí filtru v horní části stránky si *Transakce* vyfiltrujte dle *Osoby* a typu *Transakce* (např. jen příjmy).
5. Pomocí **Export transakcí** nad tabulkou se *Transakce* vyexportují do souboru a tento soubor se nabídne ke stažení.

4.4 Návrh uživatelského rozhraní

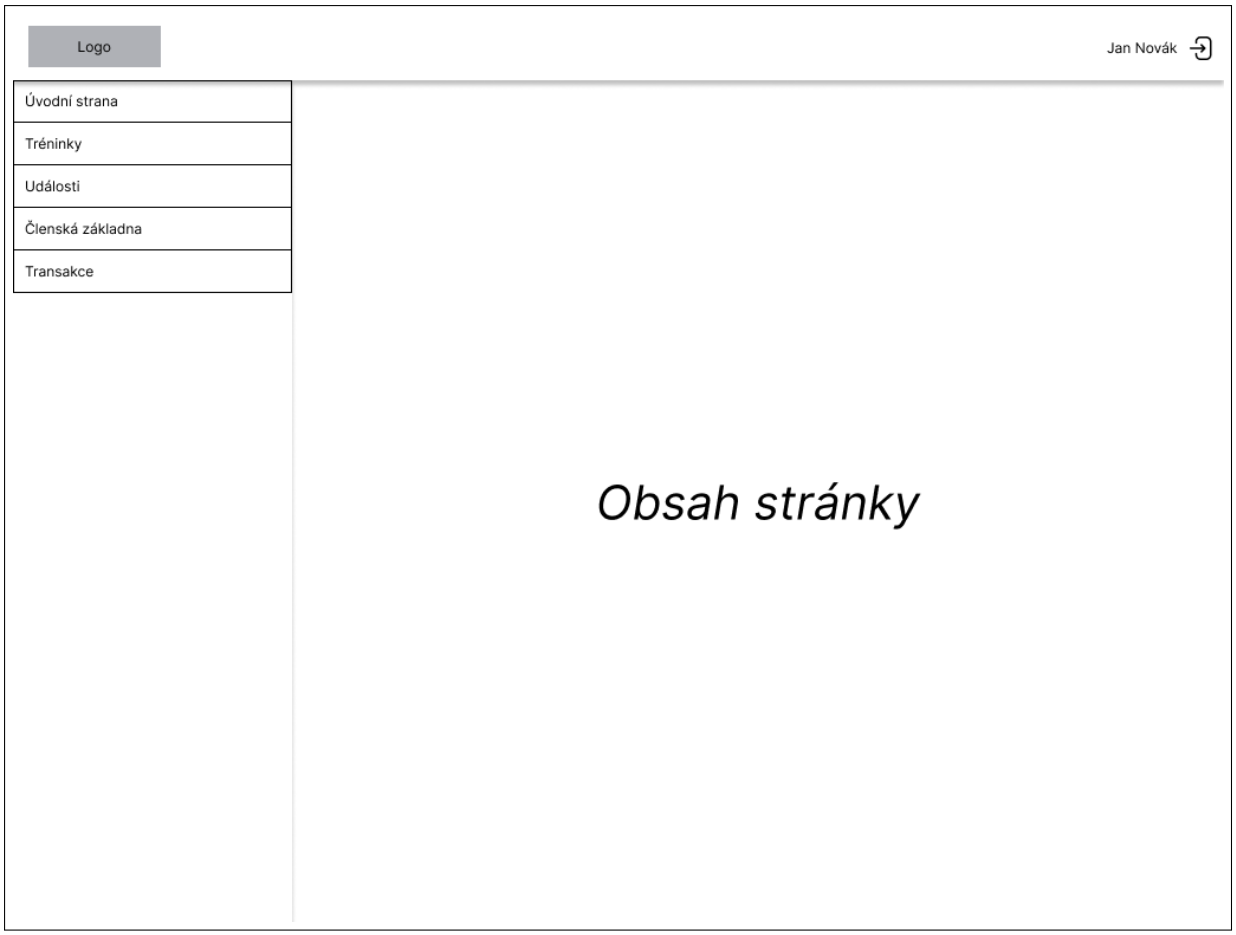
Nově vytvořený *IS* bude využívat již hotovou, volně dostupnou, šablonu AdminLTE. Osoby mající vztah s Matematicko-fyzikální fakultou UK mohou tuto šablonu znát ze systému ReCodEx.

V této kapitole budeme především popisovat rozvržení pro větší monitory (tzv. *desktop* rozložení). Mobilní rozložení bude obdobné, jen v něm budou jednotlivé prvky přeuspořádány tak, aby se vhodně vešly do menších displejů.

Popis jednotlivých rozvržení je v textu ilustrován náčrty, které poskytnou základní představu o podobě webové aplikace. Při kódování jednotlivých pohledů v implementační části budou využívány znovupoužitelné prvky dostupné v šabloně. Může tedy dojít k drobnému odchýlení se od zde prezentovaných ilustrovaných náčrtů.

Základní rozvržení bude dvousloupcové, přičemž levý sloupec bude tvořit menu s možností skrytí. Pravý sloupec bude tvořit hlavní obsah aplikace. Kromě obou sloupců se v horní části webové aplikace bude nacházet lišta obsahující logo *Organizace*, identifikaci přihlášeného *Uživatele* a možnost přihlášení.

Kromě prvního náčrtu, zobrazujícího celé rozložení aplikace, se zbývající náčrty budou zabývat výlučně obsahovou stránkou aplikace (bez levého sloupce a horní lišty).



Obrázek 4.5: Základní dvousloupcové rozvržení

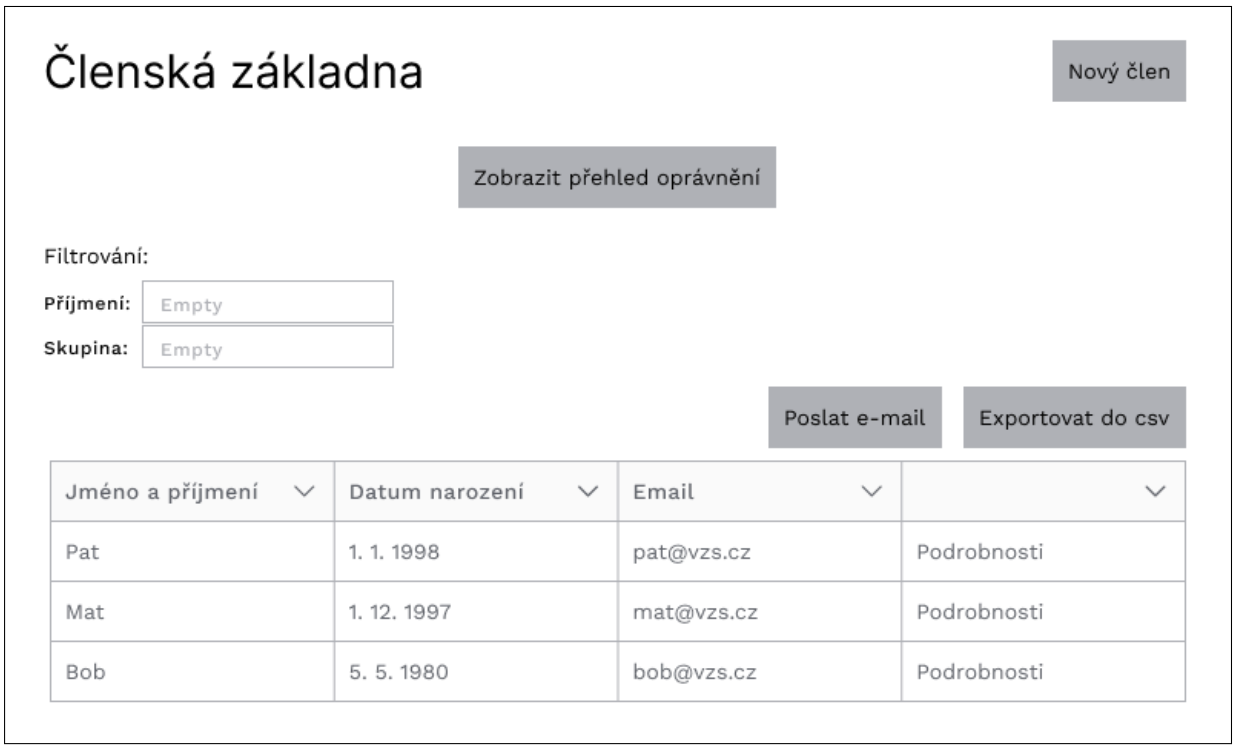
Většina pohledů budou typu seznam či detail položky. Další část pohledů bude tvořena editačními formuláři pro tvorbu nových instancí entit či úpravu stávajících.

Seznamy Seznamy *Osob*, *Událostí*, *Transakcí* a dalších entit v systému budou prezentovány tabulkovou formou. Tabulky budou mít možnost filtrace nad tabulkou podle předem vybraných parametrů. Kromě toho nad tabulkou budou tlačítka umožňující hromadné operace se seznamem (např. export dat aj.). Poslední sloupec tabulek bude tvořit prostor pro odkazy na další akce s danými položkami seznamu. Akce budou v některých případech vypsané textovou formou, v některých případech, kde to bude postačující, budou jen ikony.

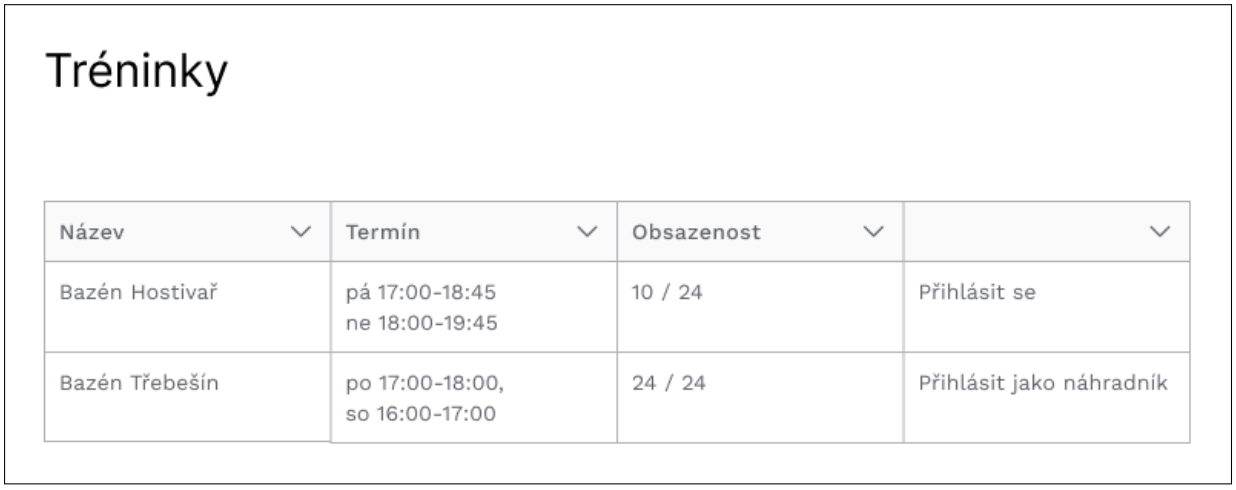
Konkrétní příklad seznamu je vidět na Obr. 4.6, kde je v seznamu zobrazena členská základna, případně na Obr. 4.7.

Detaily položek Detaily se budou využívat především pro prezentaci *Událostí*, ať už jednorázových či pravidelných. Kromě staticky prezentovaných údajů bude detail doplněn i tlačítky především s možností se přihlásit na *Událost*.

Na Obrázcích 4.8 a 4.9 jsou zobrazeny detaily položek *Událostí* s informacemi a možností přihlášení.

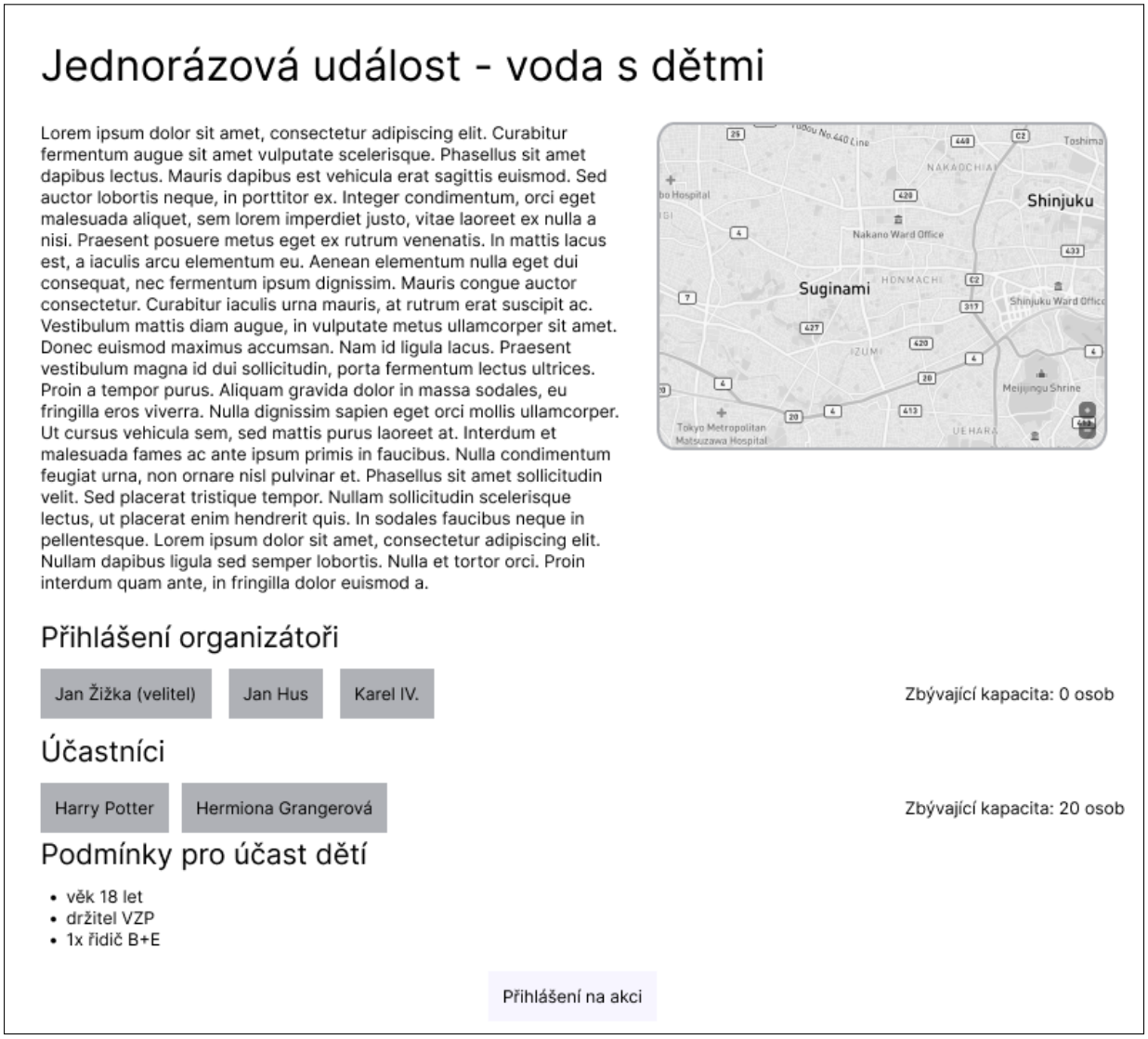


Obrázek 4.6: Seznamové rozložení – členská základna

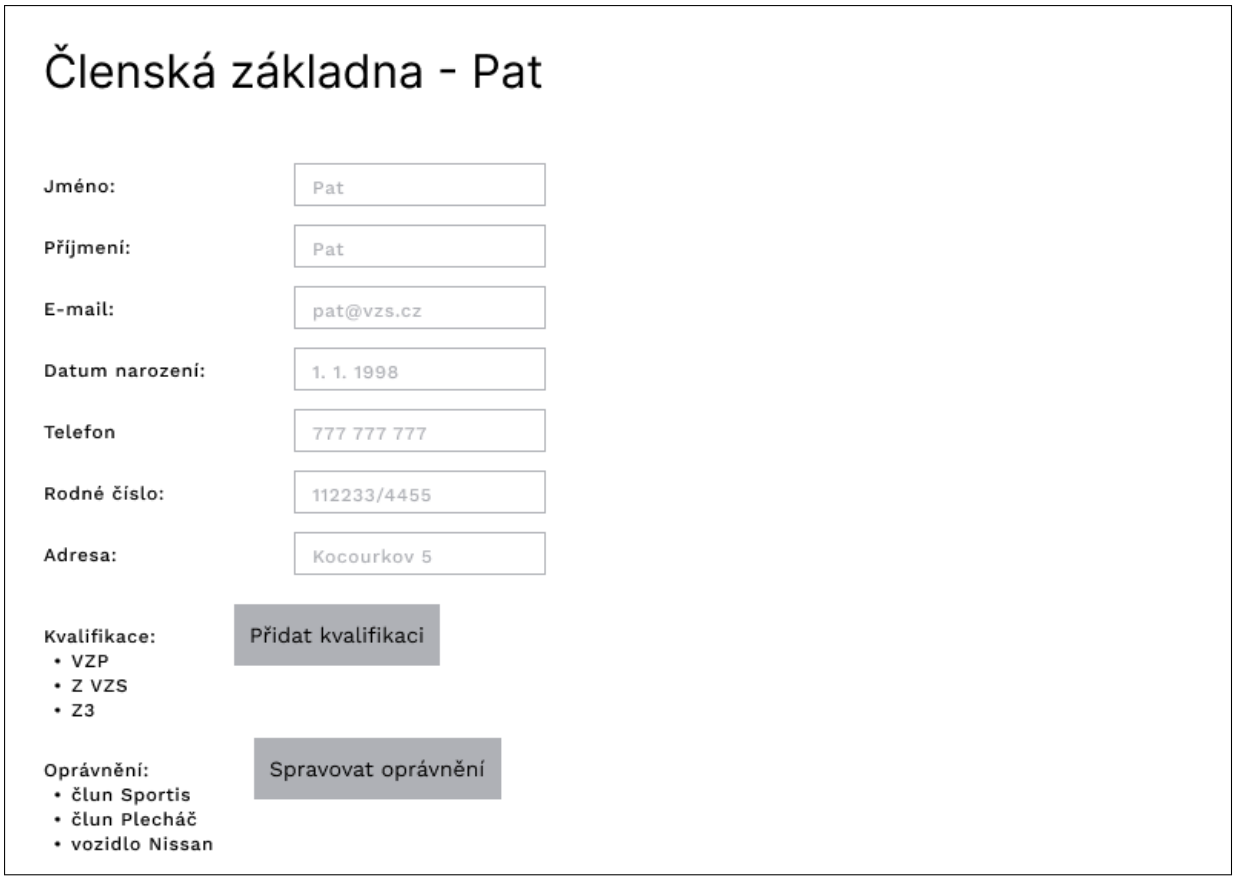


Obrázek 4.7: Seznamové rozložení – tréninky

Obrázek 4.8: Detaily položek – událost jen s organizátory (např. komerční událost)



Obrázek 4.9: Detaily položek – událost s dětmi i organizátory (např. událost pro děti)



Obrázek 4.10: Editační formulář – člen

Editační formulář Pro zakládání nových instancí událostí, členů a dalších budou využívané editační formuláře. Tyto formuláře budou mít část údajů jen pro čtení v případě, že to budou vyžadovat uživatelská práva. Příklad editačního formuláře je vidět na Obrázku 4.10.

Některé editační formuláře mohou být jen uvedeny jako seznam zaškrtávacích políček. Příkladem tohoto může být správa oprávnění u konkrétní *Osoby* viz obrázek 4.11.

Obrázek 4.11: Editační formulář – člen

5 Projektové řízení

Pro úspěšnou realizaci projektu nesmíme opomenout ani na řízení projektu samotného. Správně nastavená kritéria a kontrola postupu nám umožní případné problémy určit, tak abychom se jim přímo vyhnuli, nebo jejich řešení nestálo příliš mnoho úsilí. V závislosti na možnostech jednotlivých členů projektového týmu jsme si rozdělili práci a určili procesy, platformy a technologie, které pro řízení projektu použijeme.

Základní kontrolou postupu při tvorbě projektu pro nás budou pravidelné schůzky členů projektového týmu, ze kterých pořizujeme interní zápis. Výsledky a postup poté konzultujeme se supervizorem, abychom získali zpětnou vazbu z vnějšího prostředí.

Jako našeho konzultanta jsme si zvolili osobu s největšími znalostmi prostředí *Organizace*. Pravidelně s ním konzultujeme při návrhu a tvorbě specifikace a jeho připomínky již zapracováváme.

V následujících Sekcích 5.1–5.3 je rozepsán detailnější pohled na rozdělení práce a konkrétní platformy a technologie, které jsme vybrali, aby nám s řízením projektu pomohly.

5.1 Rozdělení práce na *IS*

Při rozdělování práce je nutné si nejprve uvědomit, že *IS* jsme navrhli jako monolitickou aplikaci. Tradiční dělení na front-end a back-end pro dělbu práce se v našem případě nemusí ukázat jako nejefektivnější způsob pro nejrovnoměrnější dělbu práce.

Rozhodli jsme se jinak – využijeme konceptu aplikací, které je možné přidávat do Django projektu. Jak jsme představili v Sekci 4.1.3, *IS* se bude sestávat z jednoho Django projektu, který bude dále rozdělen na tři aplikace. Konkrétně se jedná o:

- Users
- Members
- Events

Každá z aplikací bude mít hlavního vývojáře (garanta), který za ní zodpovídá a reviewera, který bude schvalovat veškerý vývoj a kontrolovat kvalitu. Po vzájemné diskuzi s přehlédnutím k možnostem jednotlivých členů projektového týmu jsme se dohodli, že:

- Users
  - Garant: Peter Fačko

  - Reviewer: Vojtěch Švandelík

- Members
  - Garant: Vojtěch Švandelík
  - Reviewer: Jakub Levý

- Events
  - Garant: Jakub Levý
  - Reviewer: Peter Fačko

I přes naši snahu promyslet si rozdělení aplikace a dělbu práce očekáváme, že některé části se mohou ukázat jako pracnější než zatím vypadají. V budoucnu může dojít k prohození rolí, dalšímu dělení na úrovni jednotlivých aplikací pro účely zefektivnění vývoje.

5.2 Výběr platformy pro vývoj a správu kódu

Při výběru platformy pro vývoj a správu kódu jsme se rozhodovali mezi třemi hlavními platformami: GitHub[1], GitLab[2] a Gitea[3]. Nejprve bylo nutné rozhodnout, zda potřebujeme open-source variantu, kterou je možné nasadit na vlastní server. *IS* bude vyvíjen pod MIT licencí, a proto splnění tohoto požadavku nemá pro náš projekt žádný význam. Můžeme tak ze seznamu smazat Gitea, která nemá žádnou veřejnou instanci. Zbývá nám rozhodnout se mezi GitLab a GitHub, GitLab disponuje veřejnou instancí a navíc existuje fakultní instance, kterou bychom jistě preferovali, pokud bychom se rozhodli pro GitLab.

Vybrali jsme ale GitHub, a to hlavně kvůli zkušenostem s tímto prostředím, kde se vyvíjí většina open-source projektů světa.

5.3 GitHub

GitHub je momentálně jednou z nejrozšířenějších volně dostupných platforem pro kolaboraci na vývoji softwaru.

Pro *IS* použijeme jeden repozitář. Vývoj jednotlivých aplikací bude probíhat ve vlastním branch, commit do master branch proběhne přes pull request, který schválí reviewer aplikace. Master branch bude samozřejmě nastaven v režimu protected.

[1] Odkaz na oficiální stránky https://github.com.

[2] GitLab je platformou pro vývoj a správu kódu. Nabízí bezplatnou komunitní verzi, kterou je možné provozovat na vlastním serveru a komerční placenou verzi, která nabízí podporu a další funkce. Odkaz na oficiální stránky: https://gitlab.com.

[3] Gitea je dalším ze zástupců platforem pro vývoj a správu kódu. Hlavním rozdílem oproti ostatním konkurenčním řešením je, že se jedná zcela o komunitně spravovaný projekt v jehož pozadí se nenachází žádná firma. Odkaz na oficiální stránky: https://gitea.io.

Mezi další základní funkcionalitu GitHub, kterou použijeme patří GitHub Issues, kde budeme evidovat a řešit nahlášené chyby. Pro správu úkonů, které je potřeba implementovat budeme používat systém Kanban, který je přímo součástí prostředí GitHub.

5.4 Harmonogram

V této kapitole je uveden detailní harmonogram projektu. Při sestavování harmonogramu byly zohledňovány následující aspekty:

- termín určený projektovou komisí (17. ledna 2024);
- časová vytíženost členů projektového týmu během akademického roku, zkouškového období i období letních prázdnin;
- nutnost dostatečné časové rezervy na konci projektu.

Harmonogram na Obr. 5.1 jsme připravili na měsíční bázi, přičemž je určeno, co by mělo být výstupem kterého měsíce. Základní dělení harmonogramu je dle tří separátních aplikací, které *IS* tvoří. Vzhledem k rozdílnému objemu prací v jednotlivých aplikacích bude docházet k adhoc pomocem jiných členů týmu při vývoji komplikovanějších částí.

Květen úvodní práce, specifikace

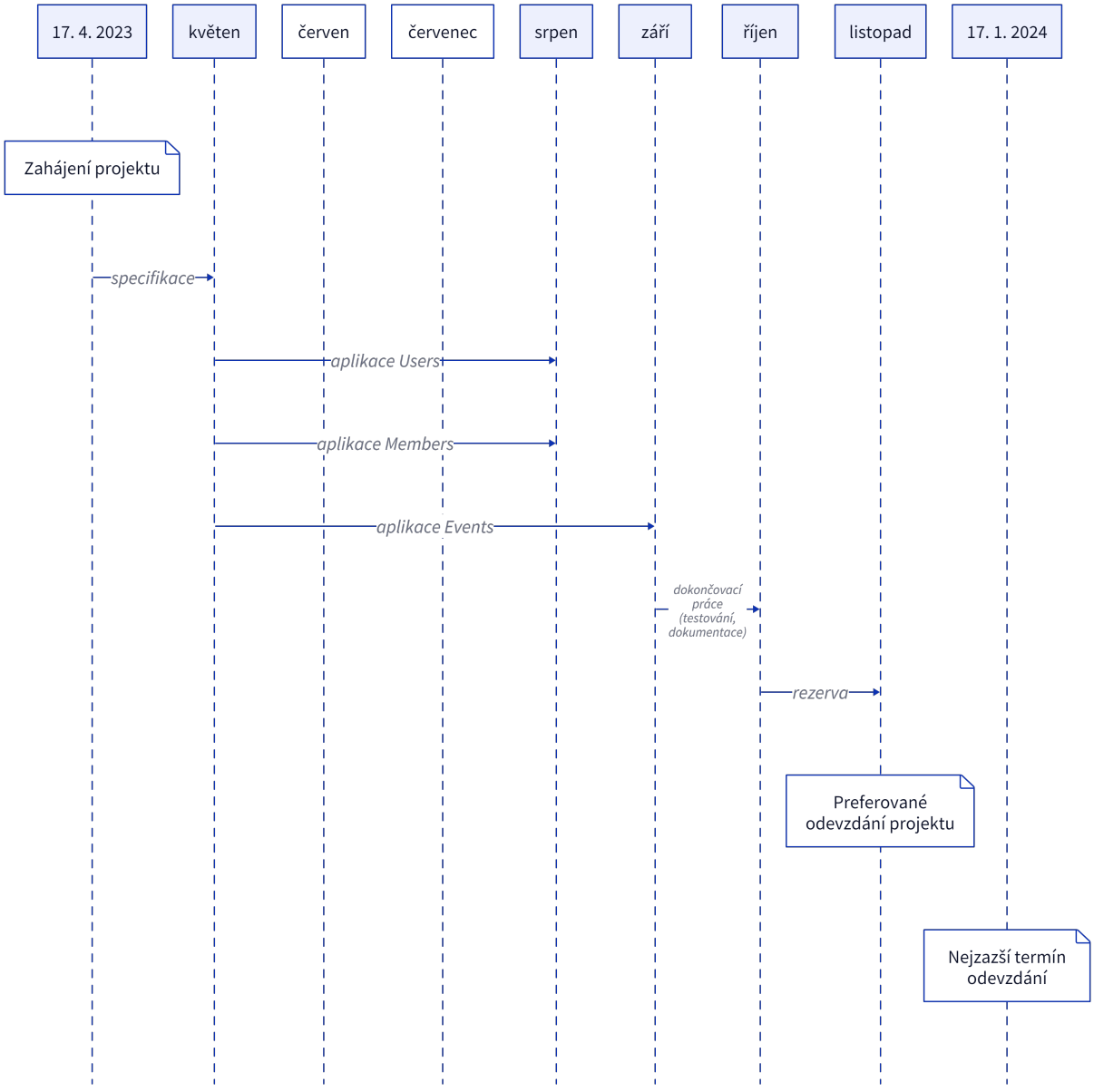
- dokončení a odevzdání specifikace
- první návrh datového modelu v Django

Červen základní struktura aplikací a jejich propojení (bez grafické podoby)

- aplikace Users: přihlašování *Uživatele*, držení uživatelské relace
- aplikace Members: zadávání *Osob* a jejich úprava (bez skupin a *Vlastností*)
- aplikace Events: zakládání *Událostí* (jen několik základních polí)

Červenec další rozšiřování především back-endové části

- aplikace Users: registrace *Uživatelů* a ověřování uživatelských *Povolení*
- aplikace Members: přidávání *Vlastností*
- aplikace Events: definování pravidelných *Událostí*, jejich struktura, přihlašování účastníků bez komplexních podmínek



Obrázek 5.1: Harmonogram

Srpen minimum viable product – hotová větší část aplikace a možnost již částečně testovat

- aplikace Users: kompletně hotové včetně grafické podoby
- aplikace Members: kompletně hotové včetně grafické podoby
- aplikace Events: pravidelné + jednorázové *Události* kompletně, přihlašování na *Události* s podmínkami, bez grafické podoby

Září dokončovací práce

- aplikace Users: vylepšování, oprava chyb
- aplikace Members: vylepšování, oprava chyb
- aplikace Events: dokončení chybějících částí, doplnění grafické podoby

Říjen dokončovací práce

- dokončení tvorby zdrojového kódu
- uživatelské testování *Systému*
- tvorba dokumentace

Projekt má naplánované odevzdání na začátku listopadu 2023. To vzhledem k předepsanému termínu, který je půlka ledna 2024, tvoří rezervu necelé tři měsíce. Naším cílem je projekt odevzdat již v listopadu především z důvodu usilovnější práce na aplikaci v průběhu letních měsíců, kdy se vývoji plánujeme věnovat ve velké míře (nebudeme se muset věnovat dalším školním povinnostem).