VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

MODULARIZACE INFORMAČNÍHO SYSTÉMU PRO FESTIVAL

FESTIVAL INFORMATION SYSTEM MODULARIZATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JÁN HANES**
AUTHOR

VEDOUCÍ PRÁCE **Ing. ZBYNĚK KŘIVKA, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Hanes Ján**
Program: Informační technologie
Název: **Modularizace informačního systému pro festival**
Festival Information System Modularization
Kategorie: Web

Zadání:

1. Seznamte se s Drupal 8 a 9 a vývojem modulů pro tento CMS a migrací ze starší verze 7. Seznamte se s informačním systémem pro podporu festivalu Animefest. Nastudujte stav význačných modulů potřebných pro existující informační systém a možnost jejich rozšíření/opravy/migrace.
2. Dle pokynů vedoucího a pořadatelů festivalu navrhněte a realizujte migraci na Drupal 9.
3. Dále navrhněte a realizujte migraci/reimplementaci gamifikačního modulu.
4. Proveďte uživatelské testování modulů i celého systému. Výsledek zhodnoťte.

Literatura:

- KŘIVOHLÁVEK, Dominik. *Informační systém podporující organizaci festivalu*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. 2020-07-08. Vedoucí práce Křivka Zbyněk. Dostupné z: https://www.fit.vut.cz/study/thesis/22680/
- DOŠLÍK, Tomáš. *Gamifikace pro podporu pořádání festivalu*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. 2018-06-13. Vedoucí práce Křivka Zbyněk. Dostupné z: https://www.fit.vut.cz/study/thesis/20727/
- SIPOS, Daniel. Drupal 9 Module Development (3rd Edition). Packt Publishing, 2020.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Křivka Zbyněk, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 26. října 2021

Abstrakt

Práce studuje modularitu informačních systémů, jejich architekturu a princip vývoje modulů. Zaměřuje se hlavně na systém správy obsahu Drupal, který analyzuje v kontextu existujícího informačního systému pro správu lidských zdrojů festivalu Animefest. Navrhuje realizaci migrace existujícího systému na novou hlavní verzi systému Drupal, implementaci podpůrných modulů a nasazení nové verze systému do provozu. Druhá část názorně ilustruje vývoj modulu se zaměřením na gamifikaci a dále analyzuje tuto problematiku.

Abstract

This Bachelor thesis focuses on the modularity of information systems, their architecture, and principles of module development. The primary focus is a content management system, Drupal. This work provides comprehensive documentation of Drupal in the context of an information system for human resource management used by Animefest. It designs a migration of an existing information system to the new main version of Drupal, implementation of support modules, and deployment. The second part practically illustrates the development of a gamification module and further analyses this issue.

Klíčová slova

modularizace, informační systém, systém správy obsahu, Drupal, gamifikace

Keywords

modularization, information system, content management system, Drupal, gamification

Citace

HANES, Ján. *Modularizace informačního systému pro festival*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Zbyněk Křivka, Ph.D.

Modularizace informačního systému pro festival

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zbyňka Křivky, Ph.D Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Ján Hanes
9. května 2022

Poděkování

Tímto způsobem bych rád poděkoval panu Ing. Zbyňku Křivkovi Ph.D. za odborný dohled a konzultace při psaní této práce. Dále bych chtěl poděkovat panu RNDr. Adamu Rambouskovi, Ph.D za technickou podporu a nakonec všem uživatelům, kteří se zúčastnili testování systému.

Obsah

Kapitola 1

Úvod

Modularizace je ve světě informačních technologií běžným termínem. Části systémů by měli být jednoduše znovupoužitelné a samostatně měnitelné. Ačkoliv se tento přístup zdá být ideálním řešením, jeho implementace bývá obtížná a při nesprávném řešení se systém stává rigidním. V práci je přiblíženo modulární řešení v systému Drupal, které poskytuje pohodlné a dynamické API, na první pohled ale velice komplikované. Dokumentace Drupalu není právě přehledná a s přihlédnutím na řadu rozdílných verzí, se celistvé návody pro vývoj modulů hledají jen těžko.

Hlavním cílem této práce, je shrnout tuto rozsáhlou dokumentaci do několik lehce pochopitelných kapitol, které vznikli jak výsledek půl roční práce na vývoji modulů pro systém Drupal 9. V závěru je ukázka návrhu a vývoje konkrétního modulu, která prakticky shrnuje celou problematiku do praktického řešení a využívá všechno co Drupal pro vývoj poskytuje.

Jako produktem potřeb pořadatelů brněnského Animefestu tato práce také přibližuje problematiku gamifikace v informačních systémech, její vliv na uživatele a poskytuje popis migrace mezi Drupalem verze 7, 8 a 9. Jedná se o informační systém HRYZ využívaný pořadateli brněnského Animefestu, primárně ke sdílení informací a evidenci pořadatelů.

Kapitola 2

Obecný rozbor systémů pro správu obsahu a jejich využití

Systém pro správu obsahu (Content management system, dále jen CMS), je systém umožňující vytváření a správu obsahu webových stránek. CMS by měl vytvářet jednoduché, uživatelsky přívětivé prostředí, umožňující i technicky méně zdatným uživatelům, upravovat obsah ve svém systému.

2.1 Dělení na základě využití

CMS jde rozdělit do dvou hlavních kategorií, dle požadavků na funkcionalitu. Jde o systémy pro správu obsahu organizací (dále jen ECM) a systémy pro správu obsahu webů (dále jen WCM). ECM je primárně využíván na správu obsahu v rámci firemní infrastruktury. WCM je využíván při správě obsahu a grafické stránky webů.

2.1.1 ECM – Enterprise Content Management

ECM je všeobecný název pro řadu strategií a metod, která se využívá při správě obsahu (dokumentů) organizace. Toto zahrnuje životní cyklus Word dokumentů, Excel tabulek, obrázků a podobně. Cílem je snižování rizika, zvýšení produktivity a efektivity procesů, pomocí eliminace fyzických dokumentů.

Typické ECM řešení umožňuje

- Ukládáni obsahu do struktur a jednoduché vyhledávání
- Automatizaci často opakovaných procesů
- Sdílení dat mezi uživateli
- Optimalizace komunikace se zákazníky
- Ukládání historie změn dat

Tyto systémy bývají často vyvíjené na míru jako proprietární software, jsou však dostupné i komerční varianty, například Alfresco od Hyland Software nebo Box ECM.

2.1.2 WCM – Web Content Management

WCM řeší podobné problémy jak ECM. Na vrch ještě poskytuje nástroje na úpravu stylu zobrazování dat. Zároveň obsahuje funkcionalitu webového frameworku (routing, šablonovací systém, bezpečnostní oprávnění...). Všechny dále rozebírané systémy budou spadat do této kategorie.

2.2 Dělení na základě architektury

CMS jako běžně žádný kus softwaru, nemají jenom jednu správnou implementaci. Proto v průběhu let vznikalo mnoho různých návrhů, jak podobný systém realizovat. Každý návrh má své využití, na základě požadavků koncového zákazníka.

CMS je typicky tvořen dvěma částmi:

- Aplikace pro správu obsahu (CMA)
- Aplikace doručování obsahu (CDA)

CMA je front-endová část, vytvářející prostředí, které umožňuje správu obsahu. CDA je back-end, který přetváří obsah uložený do databáze, na funkční stránku zobrazenou koncovému uživateli.

CMS může být dělen do několika kategorií na základě vztahu a propojení CMA a CDA. Dělí se do architektur:

- Monolitická
- Oddělená (Decoupled)
- Headless

2.2.1 Monolitická architektura

Monolitická architektura se vyznačuje integrací všech součástí (databáze, CMA, šablonovací systém) přímo do back-endu. Všechny součásti jsou tak úzce propojeny, sdílejí společné zdroje a služby, co dodává systému vysokou spolehlivost za cenu nízké flexibility. Při tomhle přístupu se front-end využívá výhradně jen na zobrazování obsahu.

2.2.2 Oddělená architektura

Oddělená architektura izoluje CMA a CDA do dvou oddělených systémů. CDA v tomto případě funguje jenom jako aplikační rozhraní volané z front-endu. CMA funguje jako oddělená aplikace (typicky aplikace postavená na JavaScript-ovom rámci, například React, Angular, Vue), nezávislá na implementaci back-endu.

2.2.3 Headless architektura

Headless architektura je v mnoha ohledech podobná oddělené architektuře. Rozdílem je že headless CMS neposkytuje žádné nástroje pro doručování obsahu. Zobrazení obsahu je plně závislé na tvůrci front-endové části systému.

2.3 Motivace využívání CMS

Jednou s hlavních výhod využívání CMS je předání kontroly nad obsahem, do rukou odborníků na subjekt podnikání organizace. Eliminují potřebu využití IT oddělení na spravování obsahu. To umožňuje technickému personálu soustředit se na podstatnější technické problémy a ulehčuje ostatnímu personálu přístup do systému, bez nutnosti komplexnějšího zaškolení.

2.4 Srovnání nejvyužívanějších CMS

Wordpress

Wordpress je v dnešní době nejvyužívanějším webovým frameworkem. Je na něm postavených až 42% všech stránek. Je to open-source CMS původně určen na tvorbu blogů. Jeho modularizovatelnost a možnost vytvářet vizuální témata, umožňuje jeho využití v řadě různých projektů. Využívání nevyžaduje žádné programovací dovednosti.

Joomla

Joomla poskytuje funkce podobné Wordpressu. Kvůli menší komunitě není je výběr modulů a témat omezenější. Tento systém je ale vhodnější pro zkušené programátory. Se správnými schopnostmi může být flexibilnější, designově totiž nebyla navrhnuta pouze na vytváření blogů.

Shopify

CMS navrhnuté na tvorbu e-commerce stránek. Poskytuje vestavěnou podporu online plateb pomocí platebních bran třetích stran. Stejně jako ostatní CMS je modulární, flexibilita je však omezena, z důvodu konkrétního zaměření na e-commerce.

Kapitola 3

Analýza redakčního frameworku Drupal

Drupal je open-source redakční framework napsaný v programovacím jazyku PHP. Je postavený nad jádrem webového frameworku Symfony. Po designové stránce je navržen pro vysokou flexibilitu a jednoduchou rozšiřitelnost. Většina jeho fungování je založena na modulech, rozšířeních, které přidávají funkce nad minimalistickým jádrem systému. Drupal je v čase psaní, devátým nejpoužívanějším CMS, s více než 588tisíci aktivními webovými stránkami.

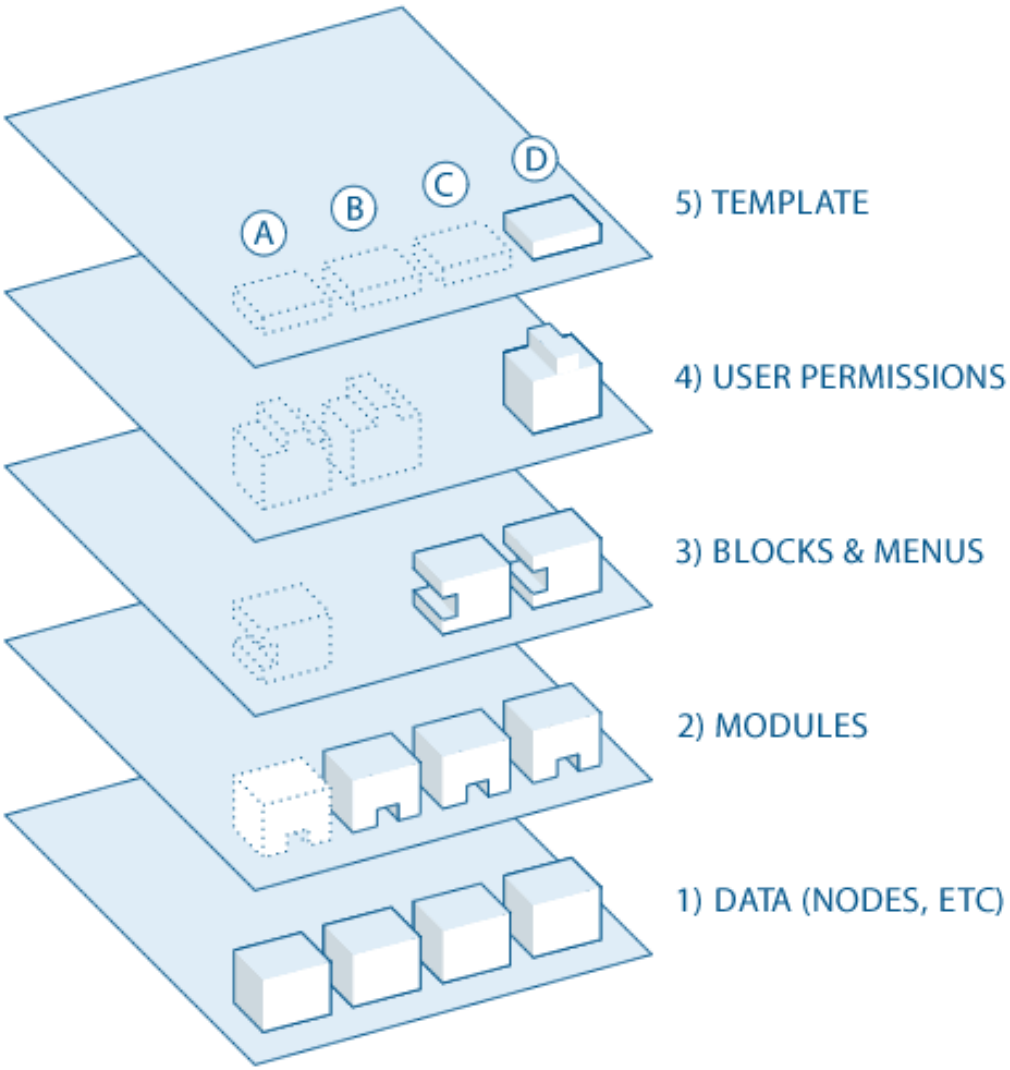
3.1 Modulární struktura jádra

Jak už bylo dříve naznačeno, Drupal obsahuje minimalistické jádro, které se stará o základní fungování webového serveru. Poskytuje služby jako zpracování požadavků, vytváření odpovědí a základní funkce zabezpečení. Na mnoho z těchto služeb využívá API jádra Symfony, nad kterým je postaven.

Velice důležitou vlastní službou Drupalu je načítání a správa modulů. Drupal dále poskytuje signalizaci událostí, primárně pomocí poněkud zastaralé technologie tzv. *hooks*. Pomocí hooks je možné spouštět funkce modulů, při určitých událostech v průběhu zpracovávání požadavků. Používání hook se snaží vývojáři v pozdějších verzích omezit, na úkor signalizace pomocí událostí a tříd fungujících jako *posluchači událostí* (anglicky event listener).

3.1.1 Vrstvy systému a tok informací

Jak je popsáno v dokumentaci Drupalu [3], při vytváření odpovědí na požadavky využívá systém jednoduchou, ale spolehlivou a pochopitelnou strukturu toku informací, založenou na vrstvách.



Obrázek 3.1: Vrstvy systém [3]

Hlavní myšlenkou je zpracování uložených dat do zobrazitelné podoby. Obrázek 3.1 dokonale ilustruje významnost modulů v systému. Proces funguje následovně:

1. Data jsou uložena jako struktury (nodes) navzájem souvisejících informací
2. Moduly načítají data, které následně upraví do zobrazitelné podoby. Typickým výstupem modulu je blok upravených dat. Moduly mohou ale zpracovávat data i opačným směrem, nebo přidávat různou funkcionalitu ostatním modulům.
3. Bloky jsou sestaveny do logických celků jako je například, stránka obsahující menu a zobrazující příspěvek a jeho komentáře. Menu, příspěvek a komentáře jsou samostatné bloky.

4. Data musí být odfiltrována na základě oprávnění uživatele. Uživatelská oprávnění jsou základním bezpečnostní prvkem a narozdíl od ostatní modulární funkcionality, jsou implementována v jádře systému.

5. Odfiltrované bloky jsou dosazeny do šablon a vykresleny jako HTML a CSS kód. Použití šablonovacího systému (v případě Drupalu se jedná o Twig), umožňuje vytváření vlastního vzhledu, pomocí šablon.

Díky této jednoznačné struktuře, je lehké najít a eliminovat případný problém v toku informací. Tato skutečnost je užitečná zvlášť při vývoji nových modulů.

3.1.2 Oprávnění a zabezpečení

Pro přístup ke konkrétním funkcím, zobrazením, nebo formulářům jsou využívány přístupová oprávnění. Drupal 9 se vyznačuje vysokou granularitou oprávnění a neexistují v něm žádné globální oprávnění. Každý modul definuje vlastní sadu statických nebo dynamických oprávnění. Jediným způsobem jak uživateli přidělit úplný přístup ke všem modulům, je povíšení daného uživatele do role *Admin*.
Uživatelské role je možné vytvářet, editovat a přidělovat jim konkrétní oprávnění. V základu existují jenom dvě role.

Administrátor

Oprávnění role není možné editovat, má implicitně přidělen přístup k celému systému. Administrační účet se vytváří při instalaci systému a nejde odstranit.

Anonym

Speciální role pro uživatele který se proti systému neověřil přihlašovacími údaji. Anonymný uživatel nemá přístup k žádné části systému, kromně přihlašovacího formuláře, pokud to není explicitně povoleno v konkrétních obsahových entitách, nebo v oprávněních.

Drupal 9 podporuje dva druhy oprávnění, statická a dynamická. Statická oprávnění jsou zpravidla deklarována v souboru `module.permissions.yml` při vytváření modulu. Je možné je využít při omezování přístupu ke konkrétním URI. Dynamická oprávnění fungují podobně, jsou ale vytvářeny na základě konfigurace nebo stavu systému, postupně za běhu.

Oprávnění je možné aplikovat i na konkrétní instance obsahových entit pomocí abstraktní třídy `EntityAccessControlHandler`. V jejím rozšíření je možné definovat libovolné požadavky pro přístup ke konkrétní instanci. Ku příkladu, uživatel může zobrazit detaily skupin, kterých je členem.

3.1.3 Caching

I když Symfony, systém který leží v jádře Drupalu, velmi rychlí, systém jako CMS může dosáhnout obrovských rozměrů. Pro zachování rychlosti jsou nutné cache. Cache je zaužívaný termín a podle definice [2] reprezentuje místo, pro uskladnění dat k urychlení pozdějšího přístupu. Jak popisuje článek zaměřený na shrnutí systému cache v Drupalu [4], cache se tvoří odlišně pro přihlášené a nepřihlášené uživatele. V jádru systému jsou dva základné moduly pro cachování.

Cache pro statické zobrazení

Stránky zobrazitelné anonymním uživatelem mohou být zálohovány celé. Takto cachované stránky obcházejí větší část vykreslovacího procesu a proto je zobrazení extrémně rychlé.

Cache pro dynamické zobrazení

Data přihlášených uživatelů jsou variabilnější, do cache se musejí ukládat pouze části stránek. Zrychluje sestavování stránek, při aktualizaci dat entit je ale důležité tuto vrstvu mazat.

3.1.4 Vykreslování a témy

Na vykreslení obsahu využívá Drupal komplexní vykreslovací pipeline[1]. Na začátku stojí požadavek, který je systémem zpracován a v případě že formát výstupu má být HTML se požadovaná data převedou do tzv. „vykreslovacího asociativního pole“. Vykreslovací pole je strukturované pole popisující data a pravidla jejich zobrazení.

```
$page = [
  '#type' => 'page',
  // 'content' is a nested render array, or child element.
  'content' => [
    'system_main' => [...],
    'another_block' => [...],
    '#sorted' => TRUE,
  ],
  'sidebar_first' => [
    ...
  ],
];
```

Správně sestavené vykreslovací pole je možné převést přímo do HTML značek.
Jakými značkami budou jednotlivé elementy vykreslovacího pole reprezentovány, závisí na tématu, které systém využívá. V Drupalu je možné vytvářet vlastní typy elementů, nebo upravovat zobrazení již definovaných. Po vykreslení je obsah odeslán jako HTML odpověď. Drupal je také implicitně připraven na zpracovávání jiných druhů komunikace, například pomocí Ajax [2] . Celý proces je lépe popsán v oficiální dokumentaci Drupalu[3].

3.1.5 Moduly nutné pro správný běh systému

S každou novou verzí, se snaží vývojáři přidávat nejvyužívanější funkcionalitu do jádra, ve formě modulů. Ačkoliv by systém fungoval i bez jakéhokoliv rozšíření, byl by použitelný pouze na vykreslování statických stránek. Vykreslování statických stránek jde proti filozofii CMS a není na to potřeba tak komplexní systém.

[1] Zřetězené zpracování, běžně používaný termín
[2] Asynchronous JavaScript and XML
[3] Drupal vykreslovací pipeline - https://www.drupal.org/docs/8/api/render-api/the-drupal-8-render-pipeline

Nodes

Drupal ukládá všechen obsah v podobě uzlů (node). Tento modul poskytuje rozšířené funkce nad uzly. Dovoluje:

- Vytvářet seznamy, hledat, třídit a upravovat uzly.
- Nastavit výchozí zobrazování obsahu.
- Konfigurovat a vytvářet nové typy obsahu, které je možné zobrazit na stránkách.

Block

Modul dovoluje vytvářet, konfigurovat a rozmísťovat vizuální bloky. Přidává administrační stránku umožňující upravovat rozložení stránky. Bez tohoto modulu by nebylo možné upravovat umístění obsahu.

Config

Modul Config umožňuje upravovat konfigurovat fungování stránky jako celku. Tato konfigurace zahrnuje kromě jiného i jazyková nastavení, název stránky a formáty zobrazovaného obsahu (např. formát času).

User

Modul podporuje registraci, přihlašování, odhlašování a správu uživatelů. Uživatelům je možné přiřazovat role, na základě kterých může zobrazovat nebo upravovat obsah.

Menu link

Umožňuje přidávat administrátorem vytvořené odkazy do menu. Bez této funkcionality by bylo menu tvořeno výhradně odkazy, které přidali programově jednotlivé moduly.

Jádro systému obsahuje mnohem víc modulů zabezpečujících různé rozšiřující, v mnoha případech i redundantní funkce. Na příkladu je demonstrováno primárně, jaká základní funkcionalita chybí systému bez jakéhokoliv rozšíření.

3.2 Přechod z Drupalu verze 8 na verzi 9

Drupal 9 narozdíl od předchozích větších verzí, nemění nějak zvlášť strukturu celého systému. Hlavními změnami byli aktualizace závislostí na LTS[4] verze a odstranění kódu, označeného jak zastaralý, již ve verzi 8. Byla poskytnuta podpora jednoduší migrace z verze 8 ale i verze 7.

[4] **L**ong **T**ime **S**upport - Verze programu s dlouhodobou podporou

Nejpodstatnějšími změnami jsou:

- Přechod ze Symfony 3 na Symfony 4.4
- Přechod z Twig 1 na Twig 2
- Nahrazení modulu SimpleTest systémem testování PHPUnit 8
- Označení *Doctrine's Simple Annotation Parser* jako zastaralý

Samotný přechod na novější verzi řeší modul Upgrade pomocí několik jednoduchých kroků.

1. Vytvoření nového, prázdného projetu pro Drupal 9 a inicializace jeho databáze
2. Instalace kompatibilních verzí modulů ze starší verze
3. Spuštění upgrade modulu na URI /upgrade v projektu verze 9
4. Výběr databáze starší verze, migrace je následně provedena automaticky

Modul Upgrade je schopný migrovat data modulů jádra a některých komunitních modulů, pokud jsou moduly ve verzi 9 dostupné a nainstalované.

3.3 Podpora vytváření nových modulů

Drupal 8 a výš, poskytuje robustní API podporující jednoduchou implementaci uživatelských modulů. API je postaveno na principech objektově orientovaného programování (OOP). Primárně používaným konceptem je kontejner služeb, který obsahuje instance všech použitelných tříd. Kontainer je přístupný po celý běh programu a přístup k němu zabezpečuje Drupal API.

Moduly mohou být použity na zobrazovaní a načítání obsahu, úpravy běhu systému, definování nových akcií nebo jakákoliv kombinace uvedených možností. Pro vývoj mnoha důležitých součástí existují dva rozdílné přístupy:

- Imperativní - Popis pomocí struktur tříd OOJ
- Deklarativní - Popis pomocí souborů YAML

3.3.1 Deklarace nového modulu

Drupal umí rozeznat moduly na základě jejich umístění a konfiguračních souborů v těchto adresářích. Pro samotnou deklaraci existence modulu je třeba vytvořit 2 soubory.

Informace o modulu

Soubor <název modulu>.info.yml slouží na konfiguraci informací o modulu, uvádí se zde informace jako název, potřebná verze Drupalu, závislosti, popis...

```
name: Shifts
description: Shifts management
package: Animefest
type: module
version: 1.4
core: 8.x
core_version_requirement: ^8 || ^9
dependencies:
  - drupal:group
php: 7.1
```

PHP funkce pro hooks

Důležitá je i existence dalšího souboru pojmenovaného jako <název modulu>.module, ve kterém je možné implementovat hooks, na základě kterých může modul zasahovat do fungování systému. Příkladem souboru může být implementace jednoduchého hooku na úpravu zobrazování bloku.

```
function <nazev_modulu>_block_view_alter(
                              array &$build,
                              BlockPluginInterface $block)
{
  if (isset($build['#contextual_links'])) {
    unset($build['#contextual_links']);
  }
}
```

Tato funkce bude volána vždy po sestavení dat bloku do pole připraveného pro vykreslování.

Po vytvoření těchto dvou souborů, je možné modul nainstalovat v administraci Drupalu, pokud systém splňuje definované podmínky. Instalací se stane modul aktivním a hook se bude spouštět při každém požadavku, který vykresluje alespoň jeden blok. Podobných hooků je dostupná celá řada. Víc informací je možné najít v dokumentaci Drupalu[5].

[5] https://api.drupal.org/api/drupal/core%21core.api.php/group/hooks/9.3.x

3.3.2 Definice oprávnění

Drupal dovoluje každému modulu definovat svá vlastní oprávnění, které jádro používá na kontrolu přístupu k obsahu. Oprávnění jsou definována v souboru <název modulu>.permissions.yml. Jediné co třeba k definování oprávnění je uvést jeho originální strojový název, uvést jeho čitelný název a popis.

```
af_shifts create shift set:
  title: 'Create a shift set'
  description: 'Allows user to create a shift set.'

af_shifts create shift:
  title: 'Create a shift'
  description: 'Allows user to create a shift.'

af_shifts edit shift set:
  title: 'Edit a shift set'
  description: 'Allows user to edit a shift set.'
```

3.3.3 Přístupy ke směřování

Nezbytnou částí vytváření modulu je definice vlastních cest pro zobrazování a zpracování dat. Drupal umožňuje k této problematice přistoupit dvěma způsoby. Jeden je založen na konfiguračním yaml souboru, druhý na rozšíření třídy *DefaultHtmlRouteProvider*.

Deklarativní přístup

Přímočařejším způsobem je definice cesty pomocí konfiguračního souboru. Jádro soubor načítá a automaticky z této konfigurace vytvoří funkční adresy, které směřují na nakonfigurované formuláře nebo funkci ovladače. Tento způsob je plně dostačující pro většinu případů užití.

```
af_shifts.configure:
  path: '/admin/config/af_shifts/configure'
  defaults:
    _form: 'Drupal\af_shifts\Form\ConfigForm'
    _title: 'Shifts module configuration'
  requirements:
    _permission: 'af_shifts configure module'

af_shifts.view_events:
  path: '/admin/config/af_shifts/events'
  defaults:
    _controller: 'Drupal\af_shifts\Controller\EventController::viewEvents'
    _title: 'Shifts event log'
  requirements:
    _permission: 'af_shifts configure module'
```

Imperativní přístup

Způsob konfigurace cest pomocí třídy se víc hodí v případě, kdy jsou cesty vázány s určitou entitou. Realizuje se zděděním třídy `DefaultHtmlRouteProvider` a implementací funkce `getRoutes(EntityTypeInterface $entity_type)`, která vrací kolekci dostupných cest. V kódu je možné si všimnout absence konkrétního URI. V tomto případě URI dodává konfigurace entity, pomocí metody `$entity_type->getLinkTemplate("<název cesty>")`. Konfigurace entit bude podrobněji popsána v další sekci.

```
class GroupRegistrationRouteProvider extends DefaultHtmlRouteProvider {

  public function getRoutes(EntityTypeInterface $entity_type) {
    $collection = parent::getRoutes($entity_type);

    if ($configure_route = $this->getConfigureRoute($entity_type)) {
      $collection->add("af_shifts.configure", $configure_route);
    }
    if ($view_events = $this->getViewEventsRoute($entity_type)) {
      $collection->add("af_shifts.view_events", $view_events);
    }
    return $collection;
  }

  protected function getConfigureRoute(EntityTypeInterface $entity_type) {
    if ($entity_type->hasLinkTemplate('configure')) {
      $route = new Route($entity_type->getLinkTemplate('configure'));
      $route
        ->setDefault('_form', 'Drupal\af_shifts\Form\ConfigForm')
        ->setDefault('_title', 'Shifts module configuration')
        ->setRequirement('_permission', 'af_shifts configure module');
      return $route;
    }
  }

  protected function getViewEventsRoute(EntityTypeInterface $entity_type) {
    if ($entity_type->hasLinkTemplate('view-events')) {
      $route = new Route($entity_type->getLinkTemplate('view-events'));
      $route
        ->setDefault(
        '_controller',
        'Drupal\af_shifts\Controller\EventController::viewEvents'
        )
        ->setDefault('_title', 'Shifts event log')
        ->setRequirement('_permission', 'af_shifts configure module');
      return $route;
    }
  }
}
```

3.3.4 Konfigurace entit

V případě potřeby ukládání dat, Drupal API poskytuje možnost vytvoření nových datových entity. Podobně jak při cestách, existují dva různé přístupy. Při entitách je už vidět patrný rozdíl ve výhodách a využitelnosti jednotlivých metod. Pro zjednodušení modulů doporučuji preferovat metodu imperativní.

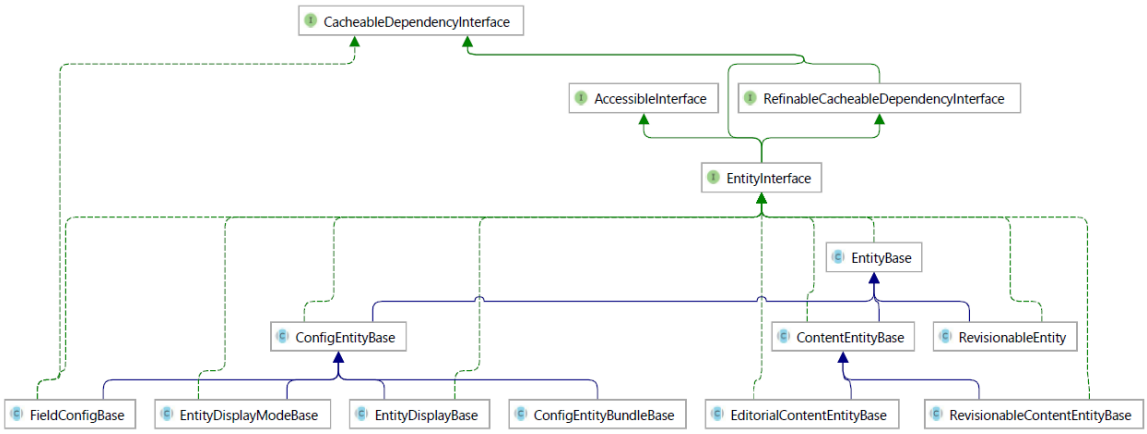
Deklarativní přístup

V souboru <název modulu>.install je možné nadefinovat databázové tabulky, které budou při instalaci vytvořeny a při odinstalaci smazány. Databázová struktura je definována pomocí konfiguračních polí, co implikuje jasnou nevýhodu. Tou je absence jakékoliv běžně využívané, vyšší funkcionality nad těmito entitami (načítání, ukládání, reprezentace pomocí třídy). Soubor pro definici tabulek může vypadat následovně.

```
<?php

function af_shifts_schema() {
  $schema = [];

  // Table of shift sets
  $schema['af_shifts_sets'] = [
    'description' => 'List of all shift sets',
    'fields' => [
      'shift_set_id' => [
        'description' => 'Shift set ID',
        'type' => 'serial',
        'not null' => TRUE,
      ],
      'shift_set_name' => [
        'description' => 'Shift set name',
        'type' => 'varchar',
        'length' => '64',
        'not null' => TRUE,
      ],
    ],
    'primary key' => ['shift_set_id'],
  ];

  return $schema;
}
```

Imperativní přístup

Na první pohled o něco komplikovanější přístup, v konečné důsledku ušetří hodně zbytečné práce. Drupal poskytuje abstraktní entitní třídy 3.2, obsahující běžnou funkcionalitu pro práci s databází.



Obrázek 3.2: UML pro rozhraní entit

Zděděnou abstraktní entitní třídu lze popsat pomocí speciálních anotací. Drupal při instalaci modulu, registruje popsané třídy a jejich strukturu překládá do databázového schématu. V databázi jsou na jeho základě následně vytvořené příslušné tabulky. Imperativní přístup také zahrnuta podpora pro vícejazyčné revidovatelné entity.

```
/**
 * @ContentEntityType(
 * id = "group_registration_request",
 * label = @Translation("Group registration request"),
 * handlers = {
 * "storage" = "GroupRegistrationStorage",
 * "list_builder" = "Drupal\group_registration\GroupRegistrationListBuilder",
 * "route_provider" = {
 * "html" = "Drupal\group_registration\GroupRegistrationRouteProvider",
 * },
 * "access" = "Drupal\group_registration\GroupRegistrationAccessControlHandler",
 * },
 * admin_permission = "administer group",
 * base_table = "group_registration_request",
 * entity_keys = {
 * "id" = "id",
 * "user_id" = "user_id",
 * "group_id" = "group_id",
 * "created" = "created",
 * },
 * links = {
 * "collection" = "/admin/group-registrations",
 * },
 * permission_granularity = "bundle"
 * )
 */
class GroupRegistrationRequest extends ContentEntityBase {
}
```

Pomocí metody `baseFieldDefinitions(EntityTypeInterface $entity_type)` lze deklarovat pole pro ukládání dat. Rodičovská třída dále poskytuje, kromě jiného, statické funkce `GroupRegistrationRequest::load($id)`, `GroupRegistrationRequest::create($array)`, instanční metody `save()` a `delete()`, které značně ulehčují manipulaci s entitami.

3.3.5 Zpracování požadavků

Důležitou součástí rozšiřování funkcionality je zpracovaní požadavků odeslaných na předem definované URI. Toto je možné dosáhnout nasměrování cesty na funkci třídy nazývané Controller. Funkce typicky přijme data z požadavku, či už se jedná o parametr v dotazu GET nebo data v těle dotazu POST. Na základě přijatých dat upraví stav systému a vrátí odpověď v libovolném formátu.

K vytváření ovladačů existuje jenom OOP přístup. Řeší se pomocí rozšíření třídy `Drupal\Core\Controller\ControllerBase`.

```
class GroupRegistrationRequestController extends ControllerBase {

  public function leaveRequest($group_id) {
    $group = Group::load($group_id);
    $currentUser = \Drupal::currentUser();
    if (!$group || !$group->getMember($currentUser)) {
      return $this->redirect('entity.group.canonical', ['group' => $group_id]);
    }
    $group->removeMember(User::load($currentUser->id()));

    \Drupal::messenger()->addMessage(t('You are no longer member of a group'));
    return $this->redirect('entity.group.canonical', ['group' => $group_id]);
  }
}
```

3.3.6 Formuláře

Formulář je typ obsahu, zobrazitelný jako samostatná stránka, nebo jako blok v rámci jiného, většího kontextu. Přístup k vytváření formulářů je čistě objektově orientovaný a řeší se pomocí rozšíření třídy *FormBase*. Pro správné fungování je nutné implementovat funkce:

- getFormId(), která vrací unikátní ID instance formuláře, dále využívané při volání hooks.
- buildForm(), ve které je potřeba definovat strukturu formuláře. Konkrétně pole, jejich typy, názvy...
- submitForm(), která se vykoná po odeslání formuláře. Běžně zpracuje odeslaná data a určí cíl přesměrování odesílatele.

```
class ExampleForm extends FormBase {

  public function getFormId() {
    return "form_id";
  }

  public function buildForm(array $form, FormStateInterface $form_state) {
    $form['#title'] = t('Example title');
    $form['description'] = [
      '#markup' => '<p>' . t('Example form description') . '</p>',
    ];
    $form['submit'] = [
      '#type' => 'submit',
      '#value' => $this->t('Submit'),
      '#button_type' => 'success',
    ];
    return $form;
  }

  public function submitForm(array &$form, FormStateInterface $form_state) {
    $form_state->setRedirect('system.admin_config');
  }
}
```

Vývoj modulů obnáší využití existujícího API, která budou vysvětleny později na konkrétních případech při rozšiřování systému HRYZ.

Kapitola 4

Informační systém HRYZ a návrh jeho aktualizace

Informační systém (dále jen IS) je podle definice v encyklopedii anglického jazyka [8] „systém na sběr, udržování a poskytování informací“. IS jsou v našem kontextu chápány jako počítačové programy, no IS může existovat i ve fyzické podobě.
V této kapitole bude analyzován informační systém brněnského Animefestu HRYZ z pohledu struktury, případů užití a procesu aktualizace. Z důvodu chybějící funkcionality bylo potřebné provést i značnou inovaci.

4.1 Popis vlastností systému

HRYZ je IS, který slouží primárně pro řízení a správu lidských zdrojů brněnského festivalu Animefest. Plní při tom dvě primární role.

4.1.1 Sdílení aktualit

HRYZ je určen primárně na evidenci a sdílení informací mezi organizátoři a vedením Animefestu. Z tohoto důvodu by měl být, jednoduše pochopitelný, i pro nově registrovaného uživatele. Pro jednoduchost, má pro organizátora systém formu informačního kanálu, který zobrazuje informace od vedení.
Systém podporuje přidávání komentářů k příspěvkům a tím také přebírá částečnou funkcionalitu diskusního fóra.

4.1.2 Podpora organizace

Pro ulehčení práce organizátorů, obsahuje systém funkce pro pokročilou evidenci uživatelů a plánování aktivit.

Profil uživatele

Každý registrovaný uživatel je povinný vyplnit profil, obsahující základní informace a informace důležité pro přidělení jeho role, a zařazení do skupiny. Registrační formulář a formulář na úpravu profilu, jsou vytvořeny pomocí modulu *Webform*.

Skupiny

Je užitečné dělit uživatele do skupin, na základě jejich pracovního zařazení. Skupinu jde následně považovat za samostatný oddíl, se kterým je možné soukromě komunikovat. Funkcionalitu skupin zabezpečuje modul *Group*, který byl v nové verzi zaveden jako náhrada modulu *Organic groups*. Chybějící funkcionalita, která umožňuje uživatelům se do skupin registrovat, byla v rámci inovace implementována v modulu *Group registrations*. Další chybějící funkce, rozesílání hromadných zpráv byla implementována v modulu *Group mass-message*. Byl také přidán modul na přidávání soukromých příspěvků do skupin.

Směnový modul

Směnový modul by implementován Dominikem Křivohlávkem jako podstatná část předchozí verze systému. Dovoluje administrátorům vytvářet balíky směn, do kterých se mohou organizátoři prihlasovat. Umožňuje udržovat přehled o rozdělení práce. Podrobnější informace o tomto modulu lze nalézt v bakalářské práci Informační systém podporující organizaci festivalu[5].

4.2 Migrace na verzi 9

Migrace probíhala z důvodů zachování struktury systému, implementovaném na starší verzi 7. Prioritou bylo zachránit data týkající se příspěvků, uživatelských účtů, jejich role a rozdělení do skupin. Migrace proběhla pomocí modulu *Upgrade*, popsaném v kapitole 3.2. Všechna data, kromě přiřazení uživatelů do skupin, se podařilo úspěšně migrovat.
Relace skupiny a uživatele, bylo potřeba transformovat do nové formy, z důvodu přechodu na novější modul *Group*. Migrace byla uskutečněna pomocí jednoduchého skriptu, který načítal data ze starého systému, na základě, kterých vytvořil ekvivalentní databázi, pomocí API, které poskytuje modul *Group*. Ukázka skriptu je v příloze A.

Tabulka *og* obsahovala definice skupin. Pro migrace byly potřeba jenom data o názvech skupin (label) a unikátní identifikátor (gid) skupiny, s jehož pomocí byla implementována asociace s uživateli. Použitím gid jako indexu pole, byl řešen případ, kdy jsou novým skupinám v databázi přidělena rozdílné identifikátory.

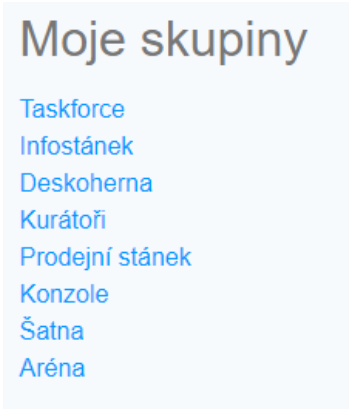
Relace byli uloženy v tabulce *og_membership*, která obsahovala identifikátor skupiny (gid) a identifikátor člena skupiny (eid). Jelikož migrace uživatelů proběhla úspěšně, bylo možné je indexovat pomocí jejich identifikátory ze staré databáze.

4.3 Využívané moduly

V této sekci budou stručně popsány některé moduly, poskytující dodatečné funkce a jejich konkrétní využití v systému HRYZ.

Views

Modul *Views* nově zařazen do jádra, poskytuje možnost vytvářet administrátorem definované pohledy na databázi. Poskytuje uživatelské rozhraní, ve kterém je možné vytvořit stránku nebo blok pomocí konkrétních dat z databáze. Data jsou získávána pomocí abstraktní formy databázového dotazu. Rozhraní umožňuje vytvářet vztahy mezi tabulkami, vybírat položky a filtrovat data.
Pomocí *Views* je vytvořen například blok *Moje skupiny*, viz obr. 4.1, zobrazující pouze skupiny, kterých je aktuálně přihlášený uživatel členem.



Obrázek 4.1: Blok skupin uživatele

Nodeaccess

Nodeaccess dovoluje definovat přístupová oprávnění k jednotlivým uzlům. To dovoluje zobrazení některých součástí menu, pouze přihlášeným uživatelům, nebo uživatelům s příslušnými oprávněními.

Flood control

Flood control umožňuje nastavovat automatické zablokování IP, při víc opakovaných neúspěšných pokusech o přihlášení. Zablokované adresy mohou být odblokovány manuálně pomocí rozhraní modulu. Modul je využíván primárně na prevenci útoků hrubou silou (brute-force attack), kdy se útočník snaží získat heslo zadáváním kombinací vstupů.

Mass Contact

Modul slouží na rozesílání hromadných e-mailů uživatelům na základě jejich role nebo s novým rozšířením i příslušenství ve skupině. Prozatím využíváno jako nejvýhodnější metoda komunikace v rámci skupin.

Field validation

Rozšířené možnosti validace entit poskytuje modul *Field validation*. Administrátor může vytvořit soubory pravidel pro entitu, podle kterých se vyhodnocují formuláře, vytvářející nebo upravující danou entitu.

Group

Modul *Group* je nezbytný pro podporu organizace lidí. Dovoluje třídit organizátory do skupin, v tomto případě podle pracovní pozice. Pro skupiny je možné definovat vlastní interní role a oprávnění pro přístup k obsahu. V původním systému plnil podobnou funkci modul *Organic Groups*, který nemá dostupnou stabilní verzi pro Drupal 9. Směnový modul aktivně využívá zařazení uživatelů do skupin, při povolování registrace do směn.

Webform

Velmi rozsáhlý modul pro podporu vytváření formulářů, dovoluje formuláře nejen vytvářet ale i velice jednoduše editovat, strukturovat a ověřovat jejich pole. Takto vytvořený formulář

je pak systémem chápán jako obsah, který lze nejen vkládat na stránky, ale je možné editovat i jeho přístupová oprávnění. *Webform* poskytuje také velké množství propojení s ostatními moduly.

Views send

Modul umožňující vytvářet vlastní mailové seznamy uživatelům pomocí modulu *Views*. Seznamy mohou být velmi univerzální, na využívání je ale nutná znalost modulu *Views* a zvýšené oprávnění. Je vhodný pro administrátory, nikoliv však pro technicky méně zdatné vedoucí.

4.4 Nově implementované moduly

Systém HRYZ neobsahoval všechnu chybějící funkcionality. Ta byla implementovaná v samostatných modulech.

Reset

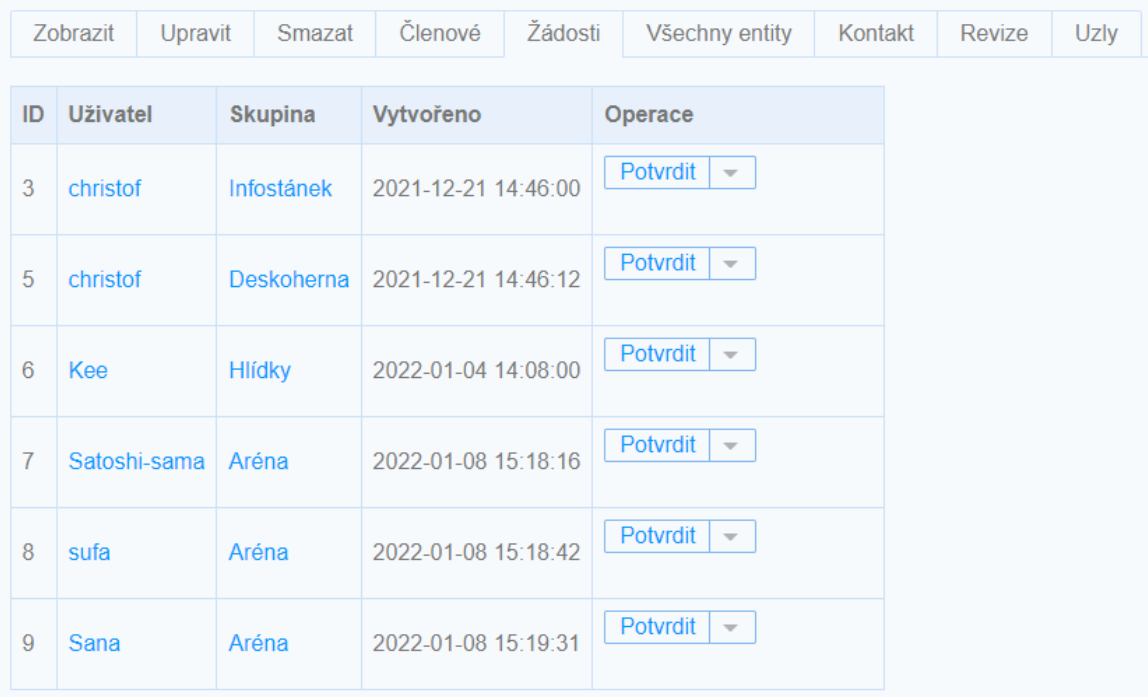
Modul poskytuje možnost, dostat systém do stavu před začátkem přípravy dalšího ročníku festivalu. Odkaz na formulář, zodpovědný za vykonání funkce, přidává do konfiguračního menu administrace pod název *Reset*. Formulář umožňuje odstranit uživatele z vybraných skupin, resetovat pole indikující účast na festivalu a exportovat členství v skupinách do CSV souboru.

Registrace do skupin

O něco komplikovanější, ovšem stále velice jednoduchý modul, přidávající chybějící možnost, podávat žádost o přidání do skupiny vytvářených pomocí modulu *Group*. Modul definuje novou entitu *Group registraton request*, která uchovává identifikátory uživatele a skupiny, do které má zájem se přidat. Dále také ukládá časové razítko podání požadavku. O přidání je možné požádat pomocí odkazu na stránce skupiny. Po kliknutí je vytvořena nová žádost, kterou může správce skupiny potvrdit nebo zamítnout. Uživatel může taktéž vytvořenou žádost zrušit, nebo po přidání skupinu obdobným způsobem opustit, viz obr. 4.2.

Obrázek 4.2: Proces podávání žádostí o členství

Modul přidává novou položku se seznamem žádostí do náhledu skupiny. Zde může uživatel s oprávněním typu *administer members* spravovat žádosti.



Obrázek 4.3: Seznam žádostí o členství

Modul ošetřuje běžné případy manipulace s URI:

- Nejde podávat žádost o přidání do skupiny, které je už uživatel členem, nebo skupiny, která neexistuje.
- Pokus o opuštění skupiny, která neexistuje nezpůsobuje chybu.
- Pokus o zrušení, potvrzení nebo odmítnutí neexistujícího požadavku vytvoří upozornění, ale ne chybu.

Hromadné zprávy v rámci skupin

Modul *Mass contact* umožňuje rozesílat e-maily skupinám uživatelů, vytvořených pomocí taxonomie[1]. Ačkoliv velice užitečný ve většině situací, modulu chybí dvě specifické vlastnosti.

- Výběr kategorie na základě skupiny modulu *Group*
- Povolení odesílání zpráv na základě interních oprávnění modulu *Group*

Zkráceně, chybí propojení s modulem *Group*. Toto propojení zabezpečuje modul *Group mass-contact*.

Modul definuje nové oprávnění skupinové *send group mass message*, které umožňuje odesílání zpráv v rámci jednotlivých skupin. Členové s tímto oprávněním mají v náhledu skupiny

[1] Taxonomie v Drupalu - https://www.drupal.org/docs/user_guide/en/structure-taxonomy.html

přístup k záložce *Kontakt*, obsahující běžný formulář využívaný modulem *Mass contact*. V tomto formuláři ale není možné vybrat běžnou kategorii adresátů a funguje výhradně pro zvolenou skupinu. Protože modul plně využívá funkcí modulu *Mass contact*, následující postup komponování a odeslání zprávy se neliší od původního postupu. Rozšíření je implementováno pomocí dědění a úpravy fungování komponent původního modulu.

Zobrazení emailu v seznamu uživatelů

Pro zobrazování seznamu entit je použita třída *EntityListBuilder*. Co je v seznamu zobrazeno, závisí na modulu a celkově není možno obsah moc editovat. V seznamu uživatelů chybělo zobrazení emailových adres, které bylo nutné doplnit pomocí nového modulu upravujícího zobrazení seznamu uživatelů.

Ovládání zobrazení polí pomocí skupinových oprávnění

Při sestavování zobrazení pomocí modulu *Views* není možné ovlivnit zobrazení jednotlivých polí, pomocí oprávnění uživatele v rámci skupiny.

Modul *Views field group permissions* doplňuje tuto funkcionalitu. V rámci bakalářské práce, byl původně implementován pro rozdíl detailnosti zobrazení seznamu uživatelů skupin, pro vedoucí role. Jeho využití je mnohem univerzálnější.

Efektu je dosaženo využitím vlastnosti třídy *ViewField*, která umožňuje definovat libovolné volitelné parametry, pomocí jednoduchého formuláře. Na základě nových parametrů, se při vykreslování upraví viditelnost pole pro jednotlivé uživatele.

Obrázek 4.4: Nastavení views field permissions

Oprávnění pro zobrazení uzlů

HRYZ využívá modul *VAPN (View access per node)*, který umožňuje konfigurovat přístup ke každé instanci obsahové entity samostatně, jednoduše z uživatelského rozhraní. Modul *Views* tato omezení nerespektoval a v seznamech zobrazoval náhledy obsahu, uživatelům kteří k němu neměli neměli přístup. Nově implementovaný modul *Views node access* tento problém odstraňuje, tím že při vytváření zobrazení porovnává výsledky s oprávněními definovanými modulem *VAPN*.

Kapitola 5

Gamifkace systému

Pojem gamifikace byl poprvé použit v roce 2008, do širšího povědomí se dostal až ve druhé polovině roku 2010. Poté začal být rychle aplikován ve velké části odvětví informačních technologií, kde se pracuje s lidským uživatelem.

Jde o proces, který má přiblížit vykonávání činností v systému hrám. Jeho cílem je zvýšit zajímavost běžných procesů a tím zvýšit motivaci uživatele k jejich vykonávání. Gamifikace je dosahována rozšířením systému o prvky využívané ve hrách. Tyto prvky poskytují uživateli výzvy a odměny, tedy zpětnou vazbu závisející na jeho činnostech. Nejpoužívanějšími gamifikačními prvky jsou v tomto ohledu bodové systémy, úrovně, řebříčky nebo plnitelné výzvy. Systém by měl uspokojovat základní lidské potřeby po odměně, nebo soutěživosti [7].

5.1 Současné řešení

V systému existovalo gamifikační řešení, které bylo uzpůsobeno fungování na systému Drupal 7. Vzhledem k značným rozdílům mezi strukturou modulů v rozdílných verzích, je jeho opětovný návrh o poznání snazší než pokus o migraci.

Jediné co se dochovalo byl jednoduchý systém hvězdiček. Každý uživatel měl individuálně přidělenu hodnotu hvězdiček, která určovala jeho významnost v organizační struktuře. Řešení fungovalo na bázi editace uživatelského pole, šlo tedy jenom o 5ti hodnotový řebříček a chyběla mu jakákoliv komplexnost. Jeden z problémů byla chybějící zpětná vazba, uživatel se neměl možnost jednoduše dozvědět důvod jeho ohodnocení.

Nové řešení rozvíjí systém hodnocení od samotného základu, poskytuje širokou funkcionalitu a přitom dává větší kontrolu do rukou administrátorů.

5.2 Návrh gamifikačních prvků

Systém je volně inspirován systémem uživatelských úrovní a úspěchů[1] na herní platformě *Steam*. Hlavními prvky jsou:

[1] angl. Achievement = odměna za dokončení určité činnosti nebo série činností

Zkušenosti

Zkušenosti vyjadřují množství úspěšné participace na aktivitách v rámci Animefestu. Reflektují spolehlivost organizátora a mají být jak motivací pro organizátora, tak i zpětnou vazbou pro vedení festivalu.

Počet zkušeností udělovaných každému uživateli, je ovlivněn počtem jeho účastí na předešlých ročnících v roli organizátora. Takzvaný modifikátor zkušeností, je implementován z důvodu přírůstku množství zkušeností potřebných pro dosáhnutí další úrovně. Modifikátor se počítá pomocí funkce

$$M(X) = 1 + log_2(P_{org} + 1) \tag{5.1}$$

kde P_{org} je počet let kdy se organizátor prokazatelně podílel na organizaci Animefestu.

Počet let	Hodnota modifikátoru
0	1
1	2,6
2	3
3	3,4
4	3,6

Úrovně

Od získaných zkušeností se odvíjí úroveň uživatele. Úroveň může být číselné nebo slovní označení rozsahů zkušeností. Nastavení úrovní je v systému lehce konfigurovatelné a nepodléhá programově nepodléhá žádným restrikcím. Při konfiguraci je ale důležité dbát jistých zásad.

Správný postup úrovní pro optimální zapojení uživatele, by se mělo držet matematického modelu, podle kterého má množství zkušeností pro dosáhnutí dalších úrovní růst exponenciálně, na základě množství zkušeností potřebných pro dosáhnutí předešlých úrovní

$$E\,(L) \;=\; a + ba + b^2a + ... \; + b^L \tag{5.2}$$

je také možné obecně zapsat jako

$$E(L) = a\;\frac{1 - b^L}{1 - b} \tag{5.3}$$

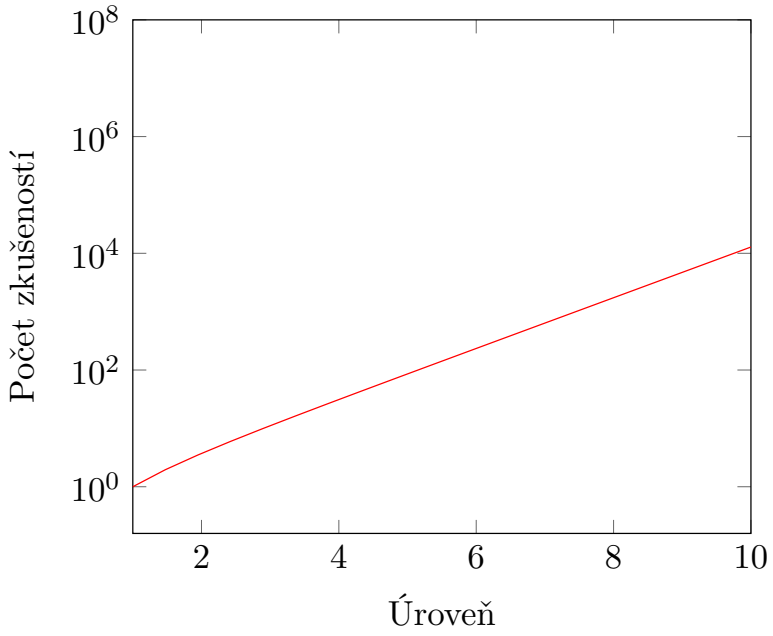
kde funkce L je pořadí úrovně a $E(L)$ vyjadřuje množství zkušeností potřebných k její dosáhnutí. a reprezentuje množství zkušeností pro dosažení první úrovně a b je modifikátor přírůstku.

V nejzákladnějším případě se atributy volí

$$\begin{aligned} a &= 1 \\ b &= e \end{aligned} \tag{5.4}$$

Tato implementace podporuje vyjadřování množství zkušeností pouze celým číslem, proto je nutné výstupy z této funkce zaokrouhlovat, nejlépe aritmeticky

$$E_r(L) = \lfloor E(L) \rceil \tag{5.5}$$



Obrázek 5.1: Růst potřebných zkušeností pro úrovně

Všechny úvahy o úrovních vycházejí ze článku zaměřeného na návrh her na stránce davideaversa.it [1].

Úspěchy

Úspěchy vyjadřují množství splnění určitých aktivit. Slouží k identifikaci uživatelů, který se pravidelně účastní stejných aktivit a jsou proto v dané činnosti zkušenější. Na rozdíl od zkušeností, které indikují celkovou zkušenost, úspěchy indikují zkušenosti v konkrétních činnostech. Za úspěchy jsou také uživateli přidělovány bonusové zkušenosti navíc. Úspěchům je možné nastavit ikonu, která se zobrazí v seznamu úspěchů na uživatelském profilu. Jde o další způsob motivace organizátorů být aktivní.

5.3 Návrh funkcionality

Aby modul splňoval požadavky pro gamifikaci musí poskytovat

- Logické administrační prostředí pro gamifikaci
- Vysokou granularitu oprávnění
- Plnou kontrolu nad gamifikačními prvky
- Zobrazování podstatných statistik pro uživatele

Bylo také třeba vyřešit tři základní otázky

- Jak budou uživatelům přidělovány zkušenosti?
- Jak budou vedoucí modul ovládat?
- Kde a jaké statistiky budou uživatelům zobrazovány?

5.3.1 Přidělování zkušeností

Původním plánem bylo implementovat systém, který by automaticky, na bázi předdefinovaných událostí v systému přiděloval uživatelům zkušenosti. Takové události by museli být plně editovatelné a z důvodu různorodosti relací entit v systému se identifikace konkrétního uživatele, kterého událost ovlivňuje, ukázala být obrovským problémem mimo rámec této práce.

Právě z tohoto důvodu byl zvolen dynamičtější, ne však plně automatický návrh. V systému existují tři hlavní zdroje zkušeností.

Aktivity

Aktivita je hromadná událost, vázaná na entitu obsahující uživatele. Po dokončení aktivity jsou všem zúčastněným přiděleny zkušenosti, ovlivněné jejich individuálními zkušenostními modifikátory. Události jsou opakovatelné a je možné libovolně upravovat seznam jejich členů. K počtu splnění aktivity se vážou úspěchy. Příkladem aktivity je účast na směně nebo členství ve skupině v rámci celého ročníku.

Úspěchy

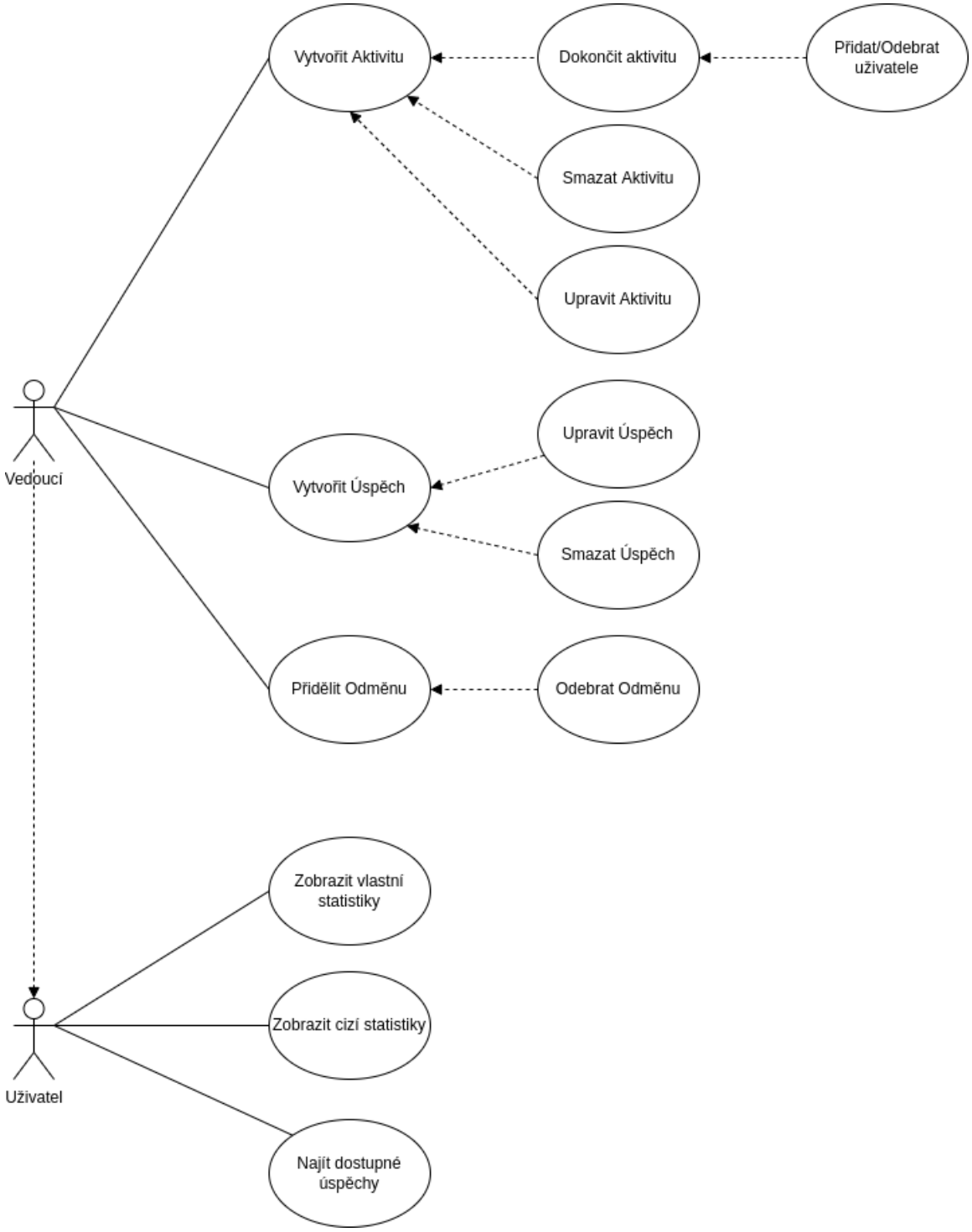
Systém pro každého uživatele zaznamenává počet splnění jednotlivých aktivit. Úspěchy mají nastavitelný počet splnění navázané aktivity, po kterém bude úspěch považován za přidělený konkrétnímu účastníkovi. Úspěchy mají nastavitelný počet přidělených zkušeností, který je ovlivněn modifikátorem zkušeností v čase splnění instance aktivity, která měla za následek přidělení úspěchu. Příkladem úspěchu může být zúčasnění se n ročnících festivalu.

Odměny

Odměny jsou individuálně přidělovány, ale na rozdíl od prostého připsání bodů jsou ovlivněny modifikátorem zkušeností a jejich důvod přidělení je zaznamenáván. Cílem je umožnit uživateli mít plný přehled o všech přidělených zkušenostech.

5.3.2 Administrace

Modul poskytuje uživatelům s dostatečnými oprávněními absolutní moc nad přidělenými zkušenostmi organizátorů. Vedoucí s maximálními oprávněními má možnost libovolně zkušenosti přidělovat, odstraňovat. Přidávat a odebírat uživatele do aktivit. Upravit uživatele v splněných aktivitách. Přidělovat a odebírat odměny. Vedoucí nemá přímou kontrolu pouze nad přidělenými úspěchy, které závisí od splněných aktivit a nad modifikátorem zkušeností, který je vázaný na uživatelské pole.



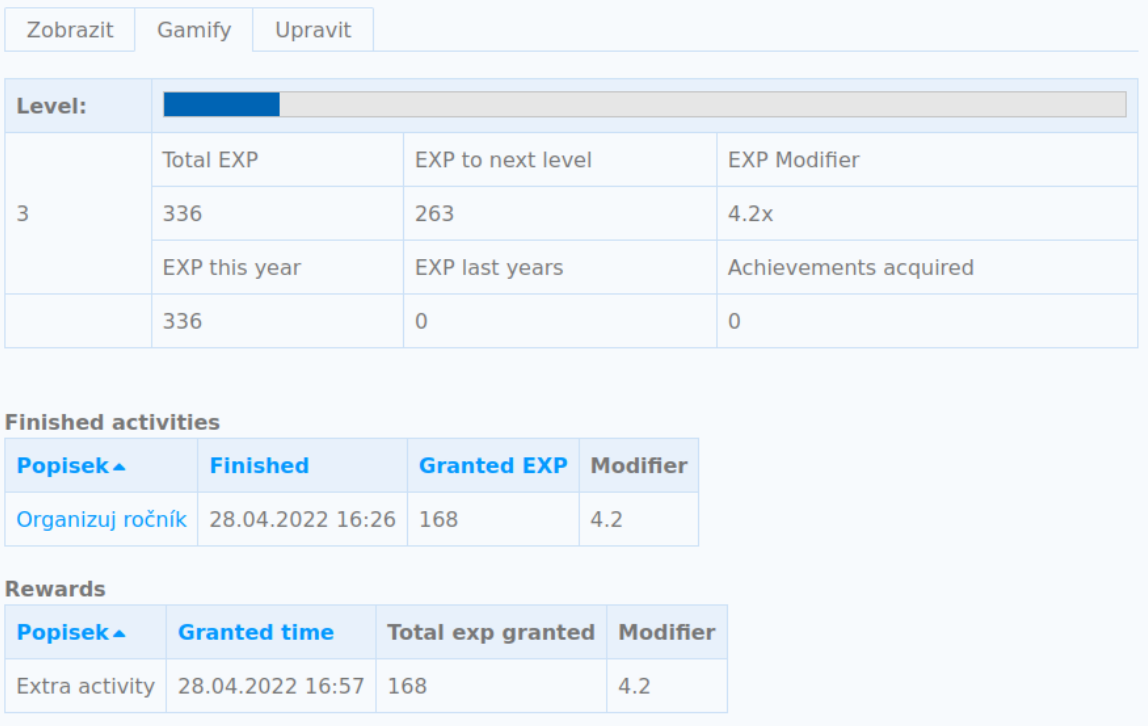
Obrázek 5.2: Use-case diagram gamifikačního modulu

Zobrazování statistik

Zobrazování statistik je řešeno novým typem zobrazovacího pole, které je umístěno v nové kartě uživatelského profilu. Pole zobrazuje

- Současnou úroveň
- Počet zkušeností
- Počet získaných úspěchu
- Hodnotu modifikátoru

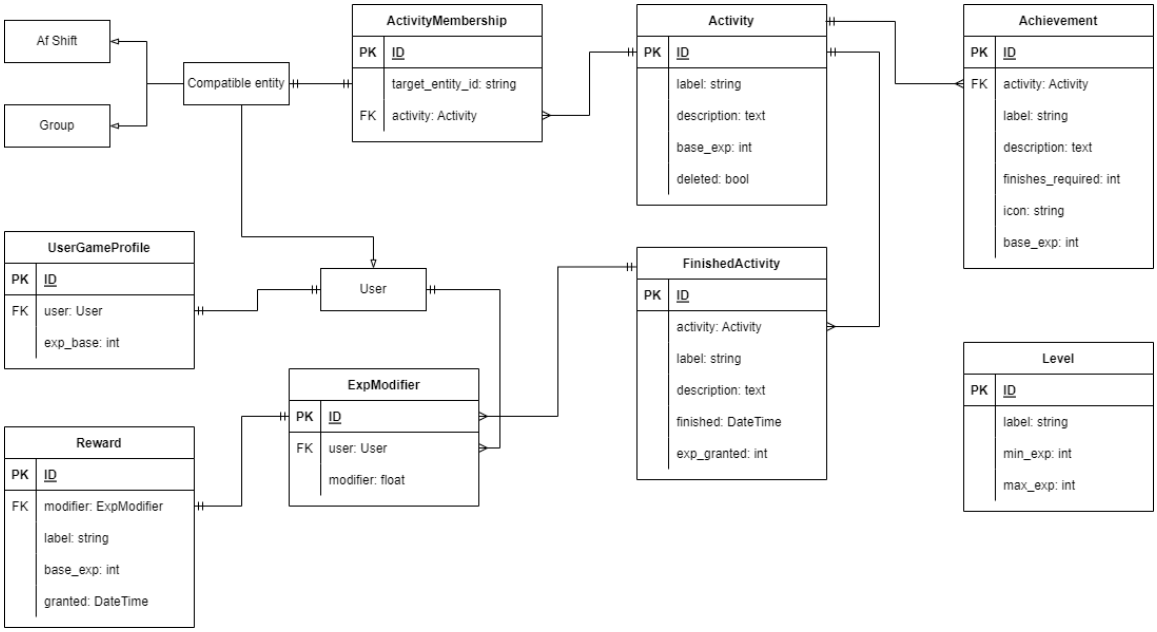
Pod statistikami jsou zobrazeny konkrétní úspěchy, dokončené aktivity a přidělené odměny. Cílem bylo poskytnout co nejkompletnější přehled o stavu uživatelů.



Obrázek 5.3: Zobrazení herního profilu

5.4 Databázové schéma

Prioritou při návrhu bylo důležité zohlednit omezení přenosu informací mezi entitami, jak je definuje Drupal. Jednou z možností bylo komplikované databázové schéma a méně komplikovaná logika, druhou jednoduché schéma a nutnost komplikovanější logiky samotných entitních tříd. Díky možnostem abstrakce, kterou entity v Drupalu poskytují a hrozbě pozdějších změn databázového schématu, jsem zvolil možnost menšího databázového schématu. To způsobilo, že některé hodnoty se místo ukládání do databáze počítají za běhu.



Obrázek 5.4: Diagram vztahů mezi entitami gamifikačního modulu

Herní profil uživatele

Herní profil měl původně skladovat větší množství statistik, ty jsou ale v současné podobě dopočítávány za běhu. V současném stavu funguje převážně jak fasáda celého systému. Jeho třída obsahuje logiku pro počítání a zobrazování statistik. V jeho detailu je také zobrazovaný grafický výpis informací pro uživatele. Před běžným ovladačem byl upřednostněn z důvodu snadné manipulace zobrazení, cest a oprávnění pro obsahové entity, kterým z pohledu systému herní profil je.

Aktivity

Aktivita je primární způsob přidělování zkušeností. Podporuje různé druhy členství, které jsou plně rozšiřitelné o jakoukoliv entitu, která je schopná poskytnout vazbu na uživatele, včetně samotného uživatele.

Aktivita ukládá její splnění v entitě *FinishedActivity*, díky které mají uživatelé přesné informace o přidělených zkušenostech. V případě potřeby odstranit aktivitu je možné vykonat archivaci. Tato možnost označí aktivitu jako archivovanou. Aktivita zmizí ze seznamu aktivit, ale vazby na úspěchy a dokončení zůstanou nedotčené. Aktivitu je následně možné odstranit úplně, co způsobí odstranění zkušeností a úspěchů přidělených uživatelům, nebo ji obnovit a opětovně používat.

Odměny

Odměna se váže specificky na jednoho uživatele a je zamýšlena pouze jako způsob přidělování zkušeností navíc.

Úrovně

Úroveň existuje ve schématu jako samostatná entita bez relací. Jedná se primárně o konfigurační entitu sloužící na volnou úpravu úrovní dle potřeby pořadatelů. Úrovně díky ní nemusí mít pouze číselné označení, ale můžou být označeny slovně například „nováček“. Potencionální poruchy konzistence vznikající překrýváním úrovní nebo mezerami, jsou řešeny na aplikační úrovni při vytváření.

Úspěchy

Úspěchy jsou přímo navázané na aktivity a nelze je přidělovat bez jejich použití. Zkušenosti přidělené úspěchem jsou počítány pomocí jejich aktivit.

Modifikátor zkušeností

Modifikátor ovlivňuje množství zkušeností připsané uživateli z aktivit, odměn nebo úspěchů. Hodnota je vytvářena automaticky při vytvoření entity přidělující zkušenosti. Funguje pouze jako prostředník a je nedostupná editaci nebo modifikaci ze strany uživatelů.

5.5 Implementace

Drupal poskytuje rozsáhlé možnosti vytváření entit a jejich interakce. Jeho API povoluje snadné vytváření CRUD[2] ovladačů, které je z velké části univerzální a automatické. Pro případ gamifikačního modulu bylo ale třeba základní metody Drupalu značně přizpůsobit.

Struktura modulu kopíruje standardní strukturu definovanou dokumentací Drupalu.

- css - nově definované css třídy
- templates - twig šablony pro zobrazovací prvky
- af_gamify.info.yml - informace o modulu
- af_gamify.install.yml - funkce spouštěné při instalaci modulu
- af_gamify.libraries.yml - deklaruje použité externí soubory, například css
- af_gamify.links.actions.yml - odkazy na akce zobrazované v rámci zobrazení entit
- af_gamify.links.menu.yml - přidává položky do menu
- af_gamify.links.task.yml - přidává karty pro přepínání zobrazení
- af_gamify.module - funkce ovlivňující běh Drupalu
- af_gamify.permissions.yml - definice nových oprávnění
- af_gamify.services.yml - vytváří služby v kontaineru
- src - používané třídy a entity

[2]CRUD - Create, read, update, delete

Entity

Entity jsou vytvářeny pomoci rozšíření abstraktní třídy *EntityBase*. Systém tyto třídy při instalaci modulu automaticky identifikuje a převede do databáze. V entitní třídě je možné definovat databázová pole, upravit chování a ovládací třídy.

Datový typ pole je určen při vytváření pomocí třídy *BaseFieldDefinition*. Databáze z těchto typů pochopí jaký typ sloupce má vytvořit.
Za zmínku stojí typ *BaseFieldDefinition::create('entity_reference')*, který se stará o vytváření vztahů mezi entitami. Dokáže vytvořit jakoukoliv relaci a automaticky řeší i dílčí tabulky relací many-to-many. V rámci tohoto vztahu je možné za běhu přistupovat přímo k referovaným entitám a v gamifikačním modulu je použit jak primární řešení vztahů mezi entitami.

Pro zjednodušení přístupu k skladovým třídám využívá každá entita PHP vlastnost *StorageProviderTrait*, která přidává statickou metodu *getStorage*

```
public static function getStorage(): ?EntityStorageInterface
{
    $entity_type_repository = \Drupal::service('entity_type.repository');
    $entity_type_manager = \Drupal::entityTypeManager();

    try {
      return $entity_type_manager->getStorage(
      $entity_type_repository->getEntityTypeFromClass(static::class)
      );
    } catch (\Exception $e) {
      return null;
    }
}
```

Metoda je nestandardní, dovoluje ale obejít náročný přístup k skladovým třídám pomocí přímého volání služby *EntityManger*.

Třídy pro přístup do databáze

Takzvané skladové třídy jsou entitní ovladače, které dovolují vytvářet specifické metody pro získávání entit z databáze. Fungují jako paralela k Symfony repozitářům. Neumožňují sice psát databázové dotazy přímo v jazyce databáze, poskytují ale jednoduché rozhraní pro vytváření těchto dotazů.

Pro zpřehlednění kódu je důležité aby každá metoda skladové třídy, vracela pouze entity, ke kterým je vázána. Ku příkladu *ActivityStorage* může no neměla by neměla vracet entity typu *Achievement*.

SqlContentEntityStorage ze které tyto třídy dědí, obsahuje standardní metody pro načítání, ukládáni a mazání entit. Velice užitečné je vytváření dotazů na základě podmínek nebo funkce pro vyhledávání pomocí hodnot polí. Jako příklad je uvedena metoda přemostující *ExpModifier* mezi *Reward* a *User* entitami ve třídě *RewardStorage*.

```
public function loadForUser(UserInterface $user): array
{
    // Build new query for controlled entity
    $query = $this->getQuery();

    // Collect ids of modifiers tied to target user entity
    $modifiers = \array_map(function ($value) {
      return $value->id();
    }, ExpModifier::getStorage()
    ->loadByProperties(['user_id' => $user->id()])
    );

    if (!$modifiers){
      return [];
    }

    // Make query collect only rewards referencting users modifiers
    $query->condition('modifier_id', $modifiers, 'IN');

    // Execute query and convert returned ids into objects
    $rewards = [];
    foreach ($query->execute() as $rewardId) {
      $rewards[] = Reward::load($rewardId);
    }

    return $rewards;
}
```

Směrování

Pro směřování je možné použít standardní *.router.yml* soubor, nebo *RouteProvider* třída. Jelikož jádrem gamifikačního modulu jsou entity byl zvolen objektový přístup. *RouteProvider* třídy jsou pro entity mnohem vhodnější, díky tomu že poskytují spoustu automatického generování cest. Běžné CRUD cesty jsou vygenerovány automaticky, pokud entita definuje vhodné formuláře a ovladače zobrazení.

RouteProvider umožňuje definovat pokročilejší cesty vázané na danou entitu jako například

```
protected function getFinishFormRoute(EntityTypeInterface $entity_type)
{
    // Check if entity defines URI template named finish-form
    if ($entity_type->hasLinkTemplate('finish-form')) {

      // Generate route object from template
      $route = new Route($entity_type->getLinkTemplate('finish-form'));

      // Set parameters necessary to build this route
      $route->setDefaults([
        '_title' => \t('Finish activity form')->render(),
        '_entity_form' => "af_gamify_activity.finish",
        'entity_type_id' => "af_gamify_activity.finish"
      ]);

      // Make route get checked by enity access controller
      $route->setRequirement('_entity_access', 'af_gamify_activity.finish');

      // Make route parameter get converted into object
      $route->setOption('parameters', [
        'af_gamify_activity' => ['type' => 'entity:af_gamify_activity']
      ]);

      return $route;
    }
}
```

která vytváří nestandardní formulář a umožňuje jeho automatické generování a ověření nestandardní operace pomocí standardního entitního ovladače přístupu.

Je tady možné také upravit standardní cesty, změnit jejich vlastnosti nebo oprávnění

```
protected function getCollectionRoute(EntityTypeInterface $entity_type)
{
    if ($route = parent::getCollectionRoute($entity_type)) {
      $route
      ->setDefault('_title', \t('Activity collection')->render())
      ->setRequirement('_permission',
      '(administer gamification)|(view activity)');

      return $route;
    }
}
```

Ovladače přístupu

Pro pokročilé ovládání přístupu k cestám a konkrétním entitám, se používají třídy děděné z *EntityAccessControlHandler*. Zděděná třída je ovladačem entity a poskytuje možnost dynamicky zkontrolovat oprávnění v závislosti na konkrétní instanci entity. Poskytuje větší volnost než statická kontrola, prováděná bez její implementace. Konkrétním případem je možnost uživatele vidět svůj herní profil, nezávisle na povoleních.

```
protected function checkAccess(
       EntityInterface $entity,
       string $operation,
       AccountInterface $account
       )
{
    // Check parent access check for explicit access decisions
    $access = parent::checkAccess($entity, $operation, $account);
    if ($access->isAllowed() || $access->isForbidden()){
      return $access;
    }

    // If parend did not decide, use your own decision rules
    if ($operation === 'view') {
      $user = \Drupal::routeMatch()->getParameter('user');
      if (User::load($account->id()) === $user) {
        return AccessResult::allowed();
      } else {
        return AccessResult::allowedIfHasPermission($account, 'view game-profile');
      }
    }

    if ($operation === 'update') {
      return AccessResult::allowedIfHasPermission($account, 'edit game-profile');
    }

    // No rule was applied
    return AccessResult::neutral();
}
```

Ovladače zobrazení

V případě přístupu k zobrazení entity se volá příslušný ovladač. Existují standardní ovladače pro zobrazení náhledu a kolekce. V gamifikačním modulu se náhled používá pouze pro zobrazení statistik uživatelského profilu a proto je zde možné najít pouze jedno přetěžení standardního ovladače. *UserGameProfileViewBuilder* se stará o přidání statistického prvku a seznamů aktivit a odměn do uživatelského profilu.

Ve větším množství se v modulu vyskytují ovladače typu *ListBuilder*, které pomáhají vykreslovat kolekce entit pro administrační účely. *ListBuilder* definuje jaké pole a operace se v seznamu zobrazí, na základě uživatelských oprávnění, nebo stavu entit.

Služba pro členství v aktivitách

V průběhu implementace se objevila výzva v podobě potřeby přidávat do aktivit různé druhy entit. V HRYZu existují dvě nejpodstatnější skupiny uživatel, pro které by mělo být možné vytvářet aktivity. Jsou to směny a skupiny. Tyto dvě entity, ovšem používají úplně jinou formu implementace. Skupiny využívají standardní Drupal rozhraní no směny toto rozhraní nevyužívají. Přístup k uživatelův v těchto entitách je velice rozdílný.

Z tohoto důvodu byla implementována služba *ActivityMembershipHandlerCollector* která funguje jako adaptér [6]. Shromáždí všechny služby implementující rozhraní *ActivityMembershipHandlerInterface* a je schopná je následně přiřadit k entitám které požadují členství v aktivitách.

Kofigurace těchto služeb v *.services.ym* by měla vypadat jako

```
af_gamify.activity_shift_membership_handler:
    class: 'Drupal\...\ActivityShiftMembershipHandler'
    tags:
      - { name: 'activity_membership_handler' }
```

Služby pak definují metodu pro identifikaci typu entity a pomocné metody pro identifikaci uživatel v členských entitách. Služby pak využívá *ActivityMembership* pro identifikaci uživatel

```
/**
* @return User[]
* @throws NoHandlerForMembershipTypeException
*/
public function getUsers(): array
{
    // Get MembershipCollector service from container
    $membershipCollector = \Drupal::getContainer()
              ->get('af_gamify.activity_membership_collector');

    // Find handler for membership type
    // Throws exception if handler does not exist
    $handler = $membershipCollector
              ->findHandler($this->get('type')->value);

    // Use handler to collect users from membership
    return $handler->getUsers($this);
}
```

Výsledkem je možnost členství skrz jakoukoliv entitu, která obsahuje relaci s uživatelem.

5.6 Stručný návod

Administrace funguje pomocí standardních zobrazení a formulářů Drupalu. Pro správné fungování je třeba

1. Nastavit rolím oprávnění pro editování gamifikačních prvků
2. Vytvořit uživatelské úrovně v administračním menu *Gamify*

Přidělit uživatelům zkušenosti můžete dvěma způsoby. Pomocí aktivit

1. Vytvořte aktivitu pomocí kliknutí na tlačítko přidat aktivitu v administračním menu
2. V seznamu operací v seznamu aktivit vyberte dokončit
3. Nastavte požadované zkušenosti, popisek a klikněte na dokončit

nebo pomocí odměn

1. Přejděte na profil uživatele
2. Otevřete záložku *EXP*
3. Při seznamu odměn, klikněte na odkaz udělit odměnu
4. Vyplňte formulář

Kapitola 6

Možnosti dalšího vylepšení

HRYZ sice svůj účel plní, prostor pro další rozšíření se najde vždy. V současném stavu je v systému možná komunikace pouze prostřednictvím mailů nebo komentářů. Implementace modulu který by umožňoval chat v rámci skupin přímo v systému, nebo by využil Discord API pro chatovací funkcionalitu by pomohl zrychlit a zefektivnit komunikaci.

Současné téma využívané systémem je generováno. HRYZ by si zasloužil vlastní téma postavené například na technologii Bootstrap pro zvýšení responzivity, popřípadě zlepšení a přizpůsobení vzhledu systému přímo pro potřeby HRYZu.

Kapitola 7

Návrh a výsledky uživatelského testování

Jelikož systém HRYZ se měl začít používat již v průběhu práce, jeho testování probíhalo postupně po celou dobu. K systému mělo po dobu 15 dní přístup přibližné 30 vedoucích a poté, byl systém zpřístupněn pro stávající a nové uživatele.

Uživatelé měli po celou dobu možnost nahlašovat problémy, které byly postupně odstraňovány a přibývala chybějící funkcionalita. Jelikož systém po dobu dvou měsíců běžel v testovacím režimu, bylo možné ho důkladně otestovat bez podrobnějších plánů, výhradně pomocí interakcí uživatelů.

7.0.1 Testování gamifikačního modulu

Uživatelské testování bylo prováděno po dokončení první verze a nasazení. Bylo zaměřeno na administrátory, kteří provádějí komplexnější interakci s rozhraním modulu. Cílem bylo zjistit přehlednost a uživatelská přívětivost administrace, splnění požadavků na funkcionalitu popřípadě detekovat chyby.

Kapitola 8

Závěr

Při narůstající potřebě robustních informačních systémů bude modularita čím dál tím důležitější součástí softwarové architektury. Webový rámec *Symfony* od ní v novějších verzích spíše upustil, čím poskytl vývojářům volnější ruku pro vynalezení vlastních řešení. Na Drupalu je zřejmé co se stane když se modularita implementuje správně ale za to dostupná dokumentace nestíhá srozumitelně popsat tak rozsáhlý systém.

Toto řešení se určitě nehodí pro každý systém, pro obecná řešení jako je jednoduchý informační systém je velice užitečná. Její problémy pramení z open-closed principu kdy je v systému třeba modul, který dělá to samé jak existující modul, jenom že trošku jinak. Pak vzniká obrovské množství volně dostupných modulů které dělají to samé a v nejhorší případě se vývojáři mohou dostat do situace kdy výstup z modulu vede do jiného modulu a ty vedou do dalšího modulu a tak dále.

Problém je částečně řešitelný systémem odposlouchávání událostí, který ale není ve velkém množství modulů implementován. Jeho správná implementace a využívání jsou velice obtížné a nehodí do každé situace.

Pro systémy, které potřebují možnost úplné modifikace, je stále vhodnější systém budovat od základu, s využitím rámce jako je *Symfony*, nebo si modularitu navrhnout podle sebe úplně od základu a podle vlastních potřeb.

Literatura

[1] Aversa, D. *GameDesign Math: RPG Level-based Progression* [online]. davideaversa.it, březen 2018 [cit. 4.5.2022]. Dostupné z: https://www.davideaversa.it/blog/gamedesign-math-rpg-level-based-progression/.

[2] Ben, L., Stacey, P. a Brien, P. *Cache* [online]. TechTarget, říjen 2021 [cit. 4.5.2022]. Dostupné z: https://www.techtarget.com/searchstorage/definition/cache.

[3] Contributors. *Drupal Overview* [online]. Drupal.org, červen 2021 [cit. 4.5.2022]. Dostupné z: https://www.drupal.org/docs/7/understanding-drupal/overview.

[4] Khurana, P. *The Fundamentals of Caching in Drupal 8* [online]. srijan.net, červen 2020 [cit. 4.5.2022]. Dostupné z: https://www.srijan.net/resources/blog/the-fundamentals-of-caching-in-drupal-8.

[5] Křivohlávek, D. *Informační systém podporující organizaci festivalu.* Brno, CZ, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/22680.

[6] Refactoring.Guru. *Refactoring.Guru - Adapter design pattern* [online]. Refactoring.Guru, 2014 [cit. 4.5.2022]. Dostupné z: https://refactoring.guru/design-patterns/adapter/.

[7] Thiebes, S., Lins, S. a Basten, D. Gamifying Information Systems – A Synthesis of Gamification Mechanics and Dynamics. In:. červen 2014.

[8] Zwass, V. *Information systems* [online]. FIT VUT v Brně, listopad 2020 [cit. 7.1.2022]. Dostupné z: https://www.britannica.com/topic/information-system.

Přílohy

Příloha A

Migrace skupin

```
<?php
# Request handling initializes container required for further calls
$autoloader = require_once 'autoload.php';
$kernel = new DrupalKernel('dev', $autoloader);
$response = $kernel->handle(Request::createFromGlobals());

# Connecting to old HRYZ database
$db = mysqli_connect(DB_SERVER, DB_USERNAME, DB_PASSWORD, DB_DATABASE);
session_start();

# Creating groups, based on old schema
$sql = "SELECT gid, label FROM `og`";
$result = mysqli_query($db, $sql);
$row = mysqli_fetch_array($result, MYSQLI_ASSOC);
$groups = [];
while ($row) {
  $groups[$row['gid']] = Group::create([
    'label' => $row['label'],
    'type' => 'bezna_skupina',
  ]);
  $groups[$row['gid']]->save();
  $row = mysqli_fetch_array($result, MYSQLI_ASSOC);
}

# Adding memebers to new groups, based on old relationships
$sql = "SELECT gid, eid FROM `og_membership`";
$result = mysqli_query($db, $sql);
$row = mysqli_fetch_array($result, MYSQLI_ASSOC);
while ($row) {
  if (isset($groups, $row['gid'])) {
    $groups[$row['gid']]->addMember(User::load($row['eid']));
  }
  $row = mysqli_fetch_array($result, MYSQLI_ASSOC);
}
```

Příloha B

Obsah přiloženého média

- xhanes00.pdf - písemná práce ve formátu PDF
- tex/ - zdrojové kódy a soubory písemné práce
- modules/ - zdrojové kódy a soubory modulů

Adresářový strom gamifikačního modulu

```
af_gamify
├── css
├── images
├── src
│   ├── Element
│   ├── Entity
│   │   ├── Access
│   │   ├── Controller
│   │   ├── Enum
│   │   ├── Form
│   │   ├── Routing
│   │   ├── Stroage
│   │   ├── Trait
│   │   └── ViewBuilder
│   ├── Exception
│   └── Plugin
│       └── ActivityMemebeshipHandler
│           └── Handlers
├── templates
└── translations
```