Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

FurryBall - správa licencí a uživatelů

Ing. arch. Šimon Steklík

Vedoucí práce: Ing. Jiří Chludil

11. května 2015

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Jiřímu Chludilovi za možnost pracovat na tomto tématu a za jeho cenné rady při psaní této práce. Poděkování patří i mé rodině za podporu po celou dobu studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 11. května 2015 .....

České vysoké učení technické v Praze

Fakulta informačních technologií

Odkaz na tuto práci

Steklík, Šimon. *FurryBall - správa licencí a uživatelů*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Tato bakalářská práce se zabývá návrhem a realizací prototypu webové aplikace pro prodej a správu licencí renderovacího softwaru FurryBall a podporu jeho uživatelů. Aplikace zahrnuje uživatelský účet, která uživateli umožňuje spravovat jeho licence, a dále administrátorskou část, sloužící ke zpracování objednávek, úpravu a analýzu dat v systému a komunikaci s uživateli.

Klíčová slova webová aplikace, informační systém, uživatelský účet, prodej a správa licencí, PHP, Symfony

Abstract

The topic of this thesis is design and inplementation of a prototype of a web application for FurryBall rendering software. The application's main purpose is sales and management of FurryBall licences and also user support. It has a user account section, where registered users can manage their licences, and administrator section, which is used to process orders, manage and analyze data and to communicate with users.

Keywords web application, information system, user account, licence sales and management, PHP, Symfony

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Bez počítačových informačních systémů se dnešní moderní společnost jen těžko obejde. Tyto systémy umožňují efektivní výměnu informací, jejich bezpečné uložení a šetří čas automatizací původně manuálně prováděných činností. Potřeby společnosti se ale rychle mění a informační systémy (jakkoliv kvalitně jsou navrženy) zastarávají. Pak je možné buď systém upravit tak, aby odpovídal novým požadavkům, nebo (pokud se požadavky změnily zásadně) je třeba vytvořit systém nový.

Firma *Art And Animation Studio* vytvářela v letech 2009 - 12 pod vedením Jana Tománka animovaný film *Kozí příběh se sýrem* a pro jeho potřeby vyvinula renderovací software FurryBall. Po skončení prací na filmu se tvůrci rozhodli nabídnout FurryBall veřejnosti. Vytvořili tedy na webu jednoduchý systém umožňující koupi permanentních i časově omezených licencí.

Postupně vznikaly nové verze softwaru FurryBall a nové typy licencí a systém pro jejich koupi se vždy ad-hoc upravil, aby umožňoval pracovat s uvedenými změnami. Zprvu jednoduchý systém a datový model na toto ale nebyly navrženy, a postupně tak docházelo k zesložitění a znepřehlednění systému, až toto dosáhlo bodu, kdy jakékoliv změny funkčnosti byly neúnosně složité a bylo rozhodnuto, že je třeba celý systém navrhnout znovu.

Moje práce se zabývá návrhem a implementací prototypu tohoto informačního systému. Nejprve jsem analyzoval požadavky zadavatele – funkce nového systému a změny oproti stávajícímu. V návrhové části jsem pak na základě této analýzy vytvořil základní strukturu systému a zvolil vhodné technologie. Poté jsem prototyp systému implementoval a připravil pro nasazení na infrastrukturu zadavatele.

Kapitola 1

Cíl práce

Cílem práce je navrhnout a implementovat prototyp webové aplikace, sloužící pro prodej a správu licencí renderovacího softwaru FurryBall a podporu jeho uživatelů. Současný systém poskytující danou funkčnost již nestačí novým požadavkům.

Prvním úkolem bude navržení a implementace uživatelského účtu – v současné době kromě nákupu veškerá komunikace s uživateli probíhá přes e-mail a přes podpůrné fórum. Uživatel nemá žádný centralizovaný přehled o tom, kolik má licencí a jaké jsou jejich vlastnosti (kdy vyprší, jaké mají identifikační číslo apod.), pokud si takový přehled sám nevytvoří. Nový uživatelský účet mu právě tento přehled poskytne. Zároveň mu umožní spravovat své licence, upravovat profilové informace apod.

Druhou částí aplikace bude administrační rozhraní, sloužící správcům systému k vyřizování objednávek, správu uživatelů a komunikaci s nimi, přehled a úpravy licencí, produktů a dalších prvků systému.

Kapitola 2

Analýza

V této kapitole se zabývám analýzou současného stavu a na základě konzultací se zadavatelem specifikuji funkční i nefunkční požadavky nového systému. Tyto požadavky poté detailněji rozepisuji do případů užití. Součástí kapitoly je také doménový model, který popisuje základní entity v systému, jejich vlastnosti a vazby mezi nimi.

2.1 Rozbor zadání

1. *Analyzujte současný stav správy licencí a uživatelů softwaru FurryBall.*

   Východiskem pro analýzu a návrh nové aplikace bude rozbor současného stavu – z jakých částí se současný systém skládá, k čemu tyto části slouží a jaké jsou mezi nimi vazby.

2. *Popište požadavky zadavatele (AAA-studio) na novou funkcionalitu webové aplikace.*

   Před samotným návrhem aplikace sestavím na základě konzultací se zadavatelem funkční a nefunkční požadavky na aplikaci. Všechny zásadní (netriviální) funkční požadavky podrobně rozpracuji do případů užití a vytvořím doménový model.

3. *Navrhněte webovou aplikaci pro správu uživatelů a licencí (schvalování, úprava licencí a uživatelských kont) včetně části pro uživatele (profil, nákup a informace o licencích atd.).*

   V návaznosti na předchozí analýzu vytvořím návrh nové aplikace. Návrh bude určitě ovlivněn použitými technologiemi, proto prvním krokem bude volba těchto technologií a jejich popis. Dalším krokem bude návrh architektury nové aplikace, popis jejích jednotlivých částí, databázového modelu a také uživatelského rozhraní.

4. *Na základě předchozích kroků implementujte prototyp webové aplikace, otestujte jej a nasaďte v infrastruktuře zadavatele.*

   Navržená aplikace bude poté implementována pomocí zvolených technologií. Nasazení na infrastrukturu zadavatele bude nejspíš probíhat už průběžně během implementace, hlavně kvůli otestování funkčnosti na zvoleném hostingu (způsob nasazení hotového prototypu aplikace bude popsán v instalační příručce). Dále také v této práci uvedu testy, kterým byla aplikace podrobena.

2.2 Definice pojmů

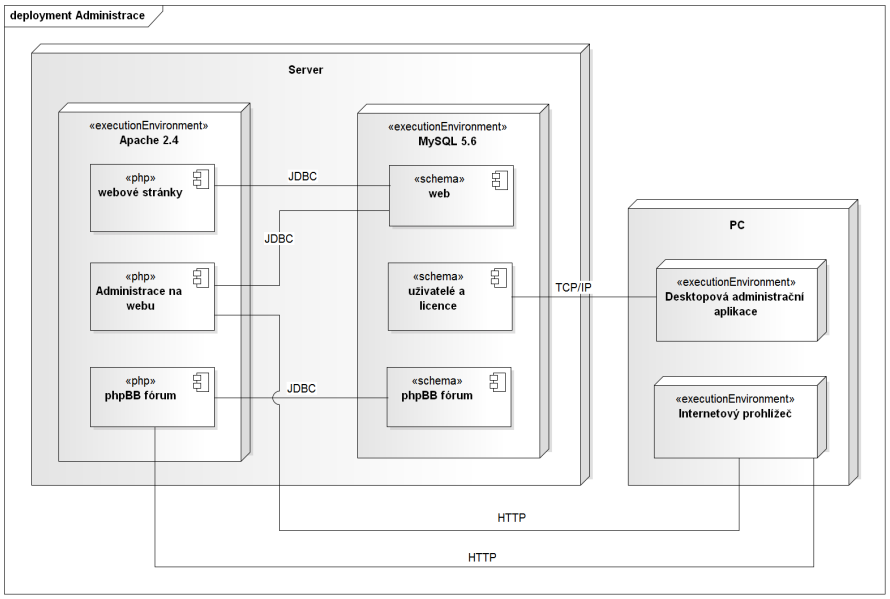
V systému existují určité entity a termíny, které jsou pro něj specifické (resp. mají v něm specifický význam). V této části se je pokusím vysvětlit.

- *HWID* – jednoznačný identifikátor konkrétního počítače (používá se primárně pro svázání počítače s licencí).
- *Licence* – entita systému identifikovaná pomocí HWID a umožňující používat FurryBall na konkrétním počítači. Tato licence je buď trvalá nebo dočasná (z toho pak vyplývají další její vlastnosti).
- *Licenční soubor* – soubor obsahující informace o licenci, který uživatel použije k aktivaci licence na konkrétním počítači.
- *FurryBall desktopový klient* – desktopová aplikace, kterou je třeba nainstalovat na počítači, kde chce uživatel renderovat pomocí softwaru FurryBall.
- *Přednostní podpora (angl. 'maintenance')* – ke každé licenci uživatel může zakoupit přednostní podporu, díky které pak má přednost při řešení problémů a další výhody. Tato podpora je dočasná a je třeba ji pravidelně obnovovat.

2.3 Současný stav

V této části popíši, jaký je současný stav systému a jak se tento počáteční stav promítne do návrhu nového řešení – které části mají být zachovány a které ne. V současnosti je celý řešený informační systém rozdělen na několik součástí:

- *webové stránky FurryBall* – web napsaný v PHP, poskytující informace o softwaru, základní nápovědu (FAQ), ukázky tvorby a umožňující nákup licencí.



Obrázek 2.1: Současné nasazení systému (z hlediska administrace)

- *webové administrační rozhraní* – toto rozhraní (taktéž napsané v PHP) slouží pro administraci webových stránek (úpravu jejich obsahu, přidávání nových stránek apod.). Dále umožňuje tvorbu a rozesílání hromadných e-mailů a nastavení některých parametrů systému.
- *podpůrné fórum* – fórum pro poskytování zákaznické podpory.
- *desktopová administrační aplikace* – aplikace nainstalovaná na několika počítačích zadavatele, která slouží administrátorům k vyřizování objednávek a analýzu dat.

Systém tedy poskytuje poměrně rozsáhlou funkčnost, jeho jednotlivé součásti jsou ale implementovány rozdílně a netvoří příliš soudržný celek. To vychází ze způsobu, jakým celý systém vznikal - na minimální jádro, u kterého se nepočítalo s rozšiřováním, se postupně nabalovaly další součásti, jak přibývalo uživatelů a funkcí.

Co v systému v současnosti nejvíc chybí, je na webu přístupný uživatelský účet - přehled uživatelových licencí s možností jejich správy (prodlužování, rušení atp.). Uživatel se v současném stavu musí sám starat o to, kolik má licencí, jaké jsou jejich vlastnosti a kdy vyprší. Přitom takový účet je už dnes na webu standardem.

Desktopová aplikace pro administraci má mnohé výhody (např. rychlost), ale její hlavní nevýhodou je dostupnost pouze na konkrétních počítačích, kde

je nainstalovaná. Webová aplikace je oproti tomu přístupná odkudkoliv, kde je k dispozici internetový prohlížeč.

Dalším zásadním problémem systému je jeho datový, resp. databázový model – stejně jako zbytek systému vznikal ad-hoc podle toho, co bylo zrovna potřeba, a ve výsledku se stal velmi nepřehledným a jakákoliv změna vyžaduje mnoho práce.

Ze všech těchto důvodů vyplývá, že není prakticky proveditelné systém pouze aktualizovat a že je třeba vytvořit systém nový. Součástí tohoto nového systému ale budou existující webové stránky FurryBall a také stávající podpůrné fórum.

2.4 Analýza požadavků

V této kapitole popisuji požadavky kladené na navrhovaný informační systém. Funkční požadavky vyjadřují, jaké funkce by měl systém vykonávat, nefunkční (obecné) požadavky popisují další nároky a omezení kladená na systém – použití určitých technologií, způsob nasazení apod.

2.4.1 Funkční požadavky

2.4.1.1 Uživatelský účet

Funkční požadavky na uživatelský účet přístupný na webu FurryBall pro registrované uživatele (s výjimkou FU1).

FU1. *Registrace uživatele* – Systém umožní registraci uživatele. Registrace je povinná pro nákup licencí.

FU2. *Automatická registrace na fóru* – Systém při registraci (FU1) automaticky registruje uživatele i na fóru (pod stejným uživatelským jménem a heslem).

FU3. *Správa účtu uživatele* – Uživatel bude mít možnost zobrazit a upravovat své profilové informace (jméno, heslo, adresa apod.).

FU4. *Nákup licence* – Systém umožní uživateli nákup licencí. Nákup bude probíhat standardním způsobem jako ve většině e-shopů - vložením do košíku, zadáním kontaktních údajů, potvrzením objednávky a zaplacením. Platba bude možná pomocí kreditní karty, PayPalu nebo převodem na účet.

FU5. *Přehled objednávek* – Uživatel bude mít přístup k přehledu svých objednávek, jejich položkám, stavu a ceně.

FU6. *Správa licencí uživatelem* – Uživatel bude mít k dispozici přehled svých licencí a bude mít možnost je spravovat. Správa bude umožňovat:

- aktivaci licence - před aktivací není možné licenci používat
- upgrade licence - převedení licence na novější verzi
- prodloužení platnosti licence (pro časově omezené licence)
- změnu HWID
- stažení licenčního souboru

FU7. *EDU registrace uživatele* – Systém umožní EDU registraci uživatele - uživatel prokáže, že je studentem/učitelem a pak může kupovat EDU licence.

FU8. *Upozornění emailem* – Systém bude zasílat uživateli (pokud o to projeví zájem) upozornění týkající se jeho licencí, např. o blížícím se konci platnosti apod.

2.4.1.2 Administrace

Funkční požadavky na rozhraní přístupné administrátorům systému.

FA1. *Správa objednávek* – Systém umožní správu objednávek, a to zejména:

- přehled a úpravu objednávek
- vytváření a mazání objednávek
- filtrování
- přehled položek objednávky
- zpracování objednávky – schválení objednávky (vždy manuální) a automatická úprava uživatelových licencí

FA2. *Správa uživatelů* – Systém umožní správu uživatelů, a to zejména:

- přehled a úpravu uživatelských dat
- vytváření a mazání uživatelů
- filtrování
- přehled uživatelových licencí
- posílání hromadných e-mailů

FA3. *Správa standardních licencí* – Systém umožní správu standardních licencí, a to zejména:

- přehled a úpravu licencí

- vytváření a mazání licencí
- filtrování
- přehled podlicencí každé licence[1]

FA4. *Historie standardních licencí* – Systém umožní uchovávat historii licencí – kdy byla zakoupena, prodloužena, že byl proveden její upgrade apod.

FA5. *Správa zkušebních licencí* – Systém umožní správu zkušebních licencí, a to zejména:

- přehled zkušebních licencí
- filtrování

FA6. *Správa e-mailů a jejich skupin* – Systém umožní správu e-mailů a jejich skupin[2], a to zejména:

- vytváření a mazání e-mailů
- přehled a úpravu e-mailů
- filtrování
- hromadné e-maily

FA7. *Správa produktů a jejich kategorií* – Systém umožní správu produktů (nabízených licencí) a jejich kategorií a to zejména:

- vytváření a mazání produktů a jejich kategorií
- přehled a úpravu produktů a jejich kategorií
- filtrování

2.4.1.3 Komunikace s desktopovým klientem FurryBall

Funkční požadavky na komunikaci systému s desktopovými klienty FurryBall (systém odpovídá klientům).

FK1. *Vytvoření a reaktivace zkušební licence* – zkušební licence se musí před použitím aktivovat pomocí desktopového klienta – systém na základě požadavku od klienta vytvoří novou zkušební licenci, resp. reaktivuje existující.

FK2. *Záznam v logu* – klienti v pravidelných intervalech hlásí svůj stav – systém toto hlášení zpracuje a zaznamená.

[1] Každá licence může obsahovat jednu nebo více podlicencí pro různé verze FurryBall pluginu

[2] Skupina e-mailů umožňuje jejich snadné kategorizování

FK3. *Přihlášení/odhlášení odběru novinek* – uživatelé se mohou pomocí desktopového klienta přihlásit k odběru novinek – systém tyto požadavky zpracuje.

2.4.2 Nefunkční požadavky

N1. *Webová aplikace* – systém bude dostupný jako webová aplikace.

N2. *PHP* – systém bude napsán v programovacím jazyku PHP[3] verze 5.

N3. *Symfony* – systém bude napsán ve frameworku Symfony 2[4].

N4. *Integrace uživatelského účtu do stávajícího webu* – uživatelský účet bude součástí stávajícího webu FurryBall.

N5. *Jazyk* – systém bude v angličtině, ale část řešící uživatelský účet bude podporovat překlad do jiných jazyků.

N6. *Hosting* – systém bude nasazen na hosting vybraný zadavatelem – *Savana hosting* ve variantě *Savana 3000*, poskytující VPS[5] a PHP ve verzi 5.5.

N7. *Databáze* – pro persistenci dat bude použit databázový systém MySQL.

N8. *Platby kartou a přes PayPal* – pro platby kartou bude použit systém GP WebPay[6]. Pro napojení na něj i na PayPal[7] budou použity knihovny dodané zadavatelem.

2.5 Účastníci (role)

Tato kapitola obsahuje výčet typů uživatelů systému a popis jejich práv. Systém je v tomto ohledu poměrně jednoduchý – obsahuje v základu pouze klasickou trojici nepřihlášený uživatel - přihlášený uživatel - administrátor.

Pomocí tohoto rozdělení se ale definuje jen přístup k základním částem systému. Pro potřeby prodeje produktů může administrátor vytvářet vlastní skupiny uživatelů.

[3] PHP - Hypertext pre-processor, programovací jazyk

[4] framework pro tvorbu webových aplikací

[5] virtuální privátní server - vyhrazená část serveru s vlastní instalací operačního systému a služeb

[6] platební brána, www.gpwebpay.cz

[7] internetový platební systém, www.paypal.com

2.5.1 Nepřihlášený uživatel

Nepřihlášený uživatel má přístup pouze k omezené části systému – může zobrazit dostupné produkty (licence) a inicializovat nákup, ale před jeho dokončením se musí přihlásit/registrovat.

2.5.2 Přihlášený uživatel

Přihlášený uživatel má práva jako nepřihlášený uživatel, a navíc může nakupovat licence a má přístup ke svému uživatelskému účtu, kde může spravovat své licence a provádět úpravy svého profilu.

2.5.3 Administrátor

Administrátor má práva jako přihlášený uživatel, a navíc přístup k administrátorskému rozhraní, které mu umožňuje spravovat data systému (může provádět všechny akce uvedené v kapitole 2.4.1.2).

2.6 Případy užití

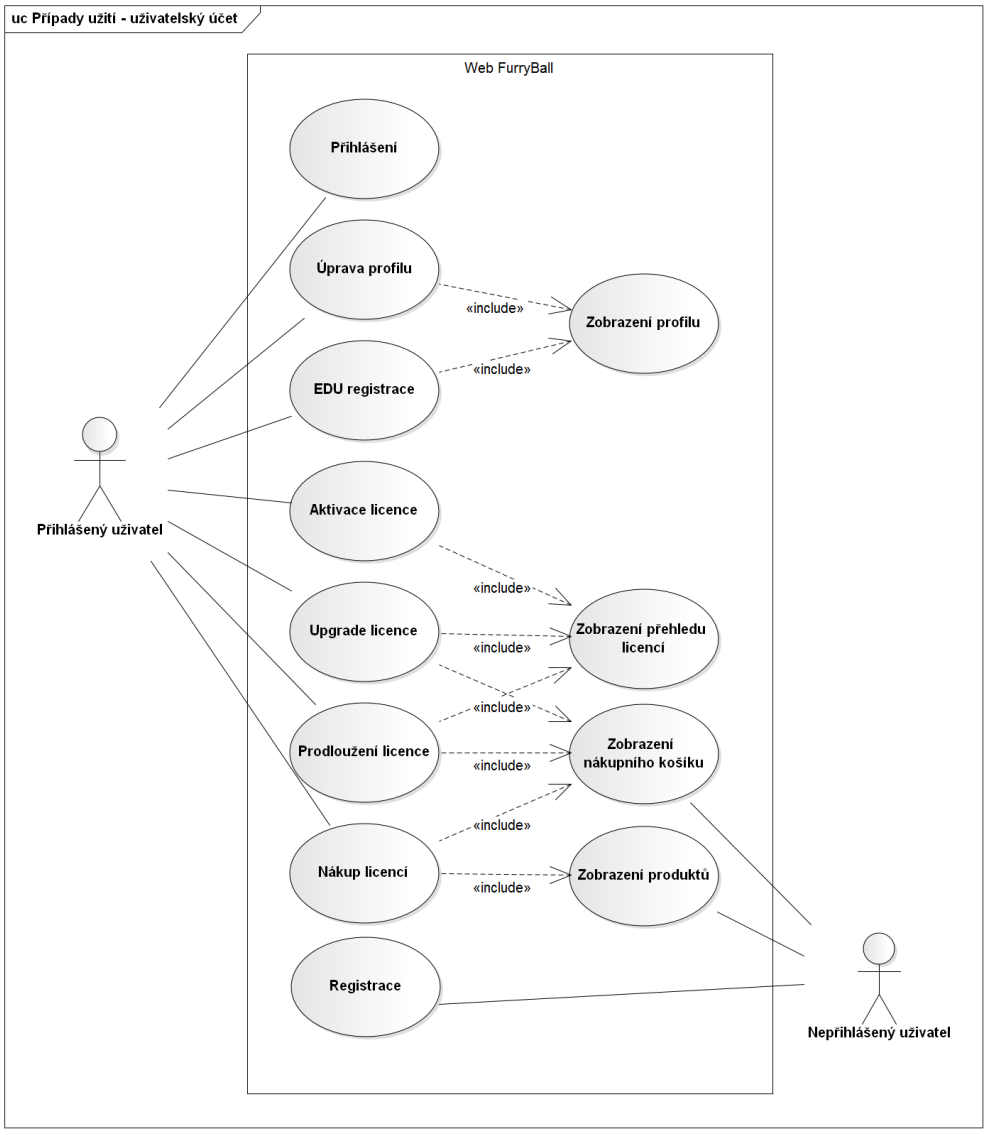
V této kapitole popisuji vybrané případy užití systému. V první části se zabývám případy užití z pohledu běžného uživatele (návštěvníka webu), druhá část popisuje práci s administrátorským rozhraním. Podrobnější scénáře uvádím jen u složitějších případů užití.

2.6.1 Uživatelský účet

UC1. *Registrace* – umožní nepřihlášeným uživatelům se registrovat

1. Scénář začíná v okamžiku, kdy se uživatel rozhodne zaregistrovat.
2. Uživateli je zobrazen formulář, který obsahuje 3 povinná pole:
   - uživatelské jméno
   - e-mail
   - heslo (+ jeho potvrzení)

   Dále obsahuje pole, která jsou pro registraci nepovinná, a buď je třeba je později vyplnit pro možnost nákupu (na toto je uživatel upozorněn), nebo slouží k upřesnění informací o uživateli.
3. Uživatel formulář vyplní a odešle.
4. Pokud je formulář vyplněn správně, je uživateli vytvořen účet, a zároveň je uživatel zaregistrován na podpůrném fóru (se stejnými přihlašovacími údaji). Pokud je formulář vyplněn chybně, pokračuje se bodem 2.
5. Uživatel je informován o úspěšné registraci a je přihlášen do svého účtu. Zároveň je mu odeslán e-mail potvrzující registraci.



Obrázek 2.2: Diagram případů užití - uživatelský účet

UC2. *Přihlášení* – umožní uživateli se přihlásit (pokud je registrován) vyplněním svého uživatelského jména/e-mailu a hesla do formuláře.

UC3. *Zobrazení profilu* – umožní přihlášenému uživateli zobrazit jeho profil – přehled jeho kontaktních informací a nastavení.

UC4. *Úprava profilu* – umožní přihlášenému uživateli měnit své profilové informace (email, adresa apod.).

1. Scénář začíná v okamžiku, kdy se uživatel rozhodne změnit své profilové informace.
2. Uživatel zobrazí svůj profil – UC3.
3. Je mu zobrazen obdobný formulář jako v UC1., bod 2., pouze bez uživatelského jména a hesla. Hodnoty jsou předvyplněny.
4. Uživatel formulář vyplní a odešle.
5. Pokud je formulář vyplněn správně, jsou změny uloženy a uživateli je zobrazen jeho profil s aktualizovanými informacemi. Pokud je formulář vyplněn chybně, pokračuje se bodem 3.

UC5. *Zobrazení přehledu licencí* – umožní přihlášenému uživateli zobrazit přehled jeho licencí se všemi informacemi o nich a s možností jejich správy.

UC6. *Zobrazení produktů (nabízených licencí)* – umožní libovolnému uživateli zobrazit přehled produktů s možností zahájení nákupu. Produkty jsou zobrazeny ve formě jejich seznamu s možností u každého specifikovat, kolik jich chce uživatel koupit.

UC7. *Zobrazení nákupního košíku* – umožní libovolnému uživateli zobrazit jeho nákupní košík s přehledem vložených produktů, celkovou cenou a možností položky v košíku upravovat.

UC8. *Nákup licencí* – umožní uživateli koupit novou licenci.

Hlavní scénář:

1. Scénář začíná v okamžiku, kdy se (přihlášený nebo nepřihlášený) uživatel rozhodne koupit novou licenci.
2. Uživateli je zobrazen přehled produktů – UC6.
3. Uživatel specifikuje, kolik a jakých licencí si přeje koupit a výběr potvrdí.
4. Vybrané licence jsou vloženy do nákupního košíku a ten je poté uživateli zobrazen.
5. V nákupním košíku uživatel může specifikovat, ke kterým licencím si přeje doobjednat přednostní podporu (pokud to daná licence umožňuje), příp. některé z košíku odstranit, a poté výběr potvrdí.

6. Pokud uživatel není přihlášen, je vyzván k přihlášení (UC2.)/registraci (UC1.). Pokračovat v nákupu může pouze přihlášený uživatel.
7. Uživateli je zobrazen formulář s údaji potřebnými pro provedení nákupu (dříve zadané informace jsou předvyplněné):
   - jméno
   - e-mail
   - adresa
   - zda je uživatel plátcem DPH[8] a příp. jeho identifikační číslo
   - s jakým softwarem bude FurryBall plugin používat
8. Uživatel formulář vyplní a odešle.
9. Pokud je formulář vyplněn správně, je uživateli zobrazen souhrn objednávky – jeho kontaktní informace, přehled položek objednávky a celková cena (rozepsaná jako základ + DPH = celková cena). Uživatel údaje zkontroluje a vybere způsob platby:
   - platební kartou
   - přes PayPal
   - převodem na účet
10. Pokud uživatel vybral platbu kartou nebo přes PayPal, je přesměrován na odpovídající platební bránu. Pokud uživatel vybral platbu převodem, je mu zobrazena informace o vytvoření objednávky a o tom, na jaký účet a s jakými údaji má peníze převést.
11. Pokud uživatel vybral platbu kartou nebo přes PayPal, je po úspěšné platbě informován o vytvoření objednávky a přesměrován na seznam svých licencí – UC5.
12. Systém odešle na uživatelův e-mail potvrzení objednávky a zároveň odešle administrátorovi e-mail s informací o vytvoření nové objednávky.

    (objednávka poté musí být schválena administrátorem – až poté dojde k vytvoření licencí)

Výjimečný scénář:

1. Pokud se v průběhu platby vyskytne chyba, je o tom uživatel informován, objednávka není vytvořena, ale nákupní košík zůstane v původním stavu.
2. Pokud je platba přes platební bránu uživatelem zrušena, objednávka není vytvořena, ale nákupní košík zůstane v původním stavu. Uživatel je informován o tom, že může nákup zrušit, nebo platbu opakovat.

[8] pouze v zemích EU

UC9. *Aktivace licence* – umožní uživateli aktivovat zakoupenou licenci.

1. Scénář začíná v okamžiku, kdy se přihlášený uživatel rozhodne aktivovat novou licenci.
2. Uživatel zobrazí přehled svých licencí – UC5.
3. Vybere licenci, kterou chce aktivovat a volbu potvrdí.
4. Zobrazí se formulář pro zadání HWID.
5. Uživatel ho vyplní a odešle.
6. Systém zkontroluje, jestli je zadané HWID platné a poté aktivuje licenci. V opačném případě se pokračuje bodem 4.
7. Uživateli je zobrazen přehled jeho licencí a je informován o aktivaci licence.

UC10. *Upgrade licence* – umožní uživateli upgrade licence. Upgrade uživatelem je možný pouze u trvalých licencí bez zakoupené přednostní podpory (licence s přednostní podporou a časově omezené licence jsou upgradovány automaticky).

1. Scénář začíná v okamžiku, kdy se přihlášený uživatel rozhodne, že chce provést upgrade licence.
2. Uživatel zobrazí přehled svých licencí – UC5.
3. Vybere licenci, kterou chce upgradovat, a volbu potvrdí.
4. Upgrade je vložen do košíku a zbytek nákupu je stejný jako UC8. od bodu 5.
5. Po zpracování objednávky je uživateli přidána nová licence novější verze – původní licence mu zůstane, ale nejde znovu upgradovat[9].

UC11. *EDU registrace* - umožní uživateli registrovat se jako EDU uživatel, díky tomu pak může koupit EDU licenci[10].

1. Scénář začíná v okamžiku, kdy se přihlášený uživatel rozhodne registrovat jako EDU uživatel.
2. Uživatel zobrazí svůj profil – UC3.
3. Vybere možnost *'registrovat jako EDU'*.
4. Zobrazí se formulář pro EDU registraci s poli (všechna povinná):
   - jméno školy
   - zda je studentem nebo učitelem
   - dokument dokazující, že je studentem/učitelem (fotografie, scan)

[9] toto chování je vyžadováno zadavatelem

[10] speciální licence, která je levnější, ale obsahuje určitá omezení

5. Uživatel formulář vyplní a odešle.
6. Pokud je formulář vyplněn správně, EDU registrace je uložena a uživatel je o tom informován.
   (EDU registraci musí schválit administrátor, až poté je platná)

2.6.2 Administrační rozhraní

Všechny případy užití v této podkapitole se týkají uživatele - administrátora.

UC12. *Zobrazení objednávek* – umožní uživateli zobrazení přehledu (seznamu) objednávek, který bude u každé objednávky obsahovat:

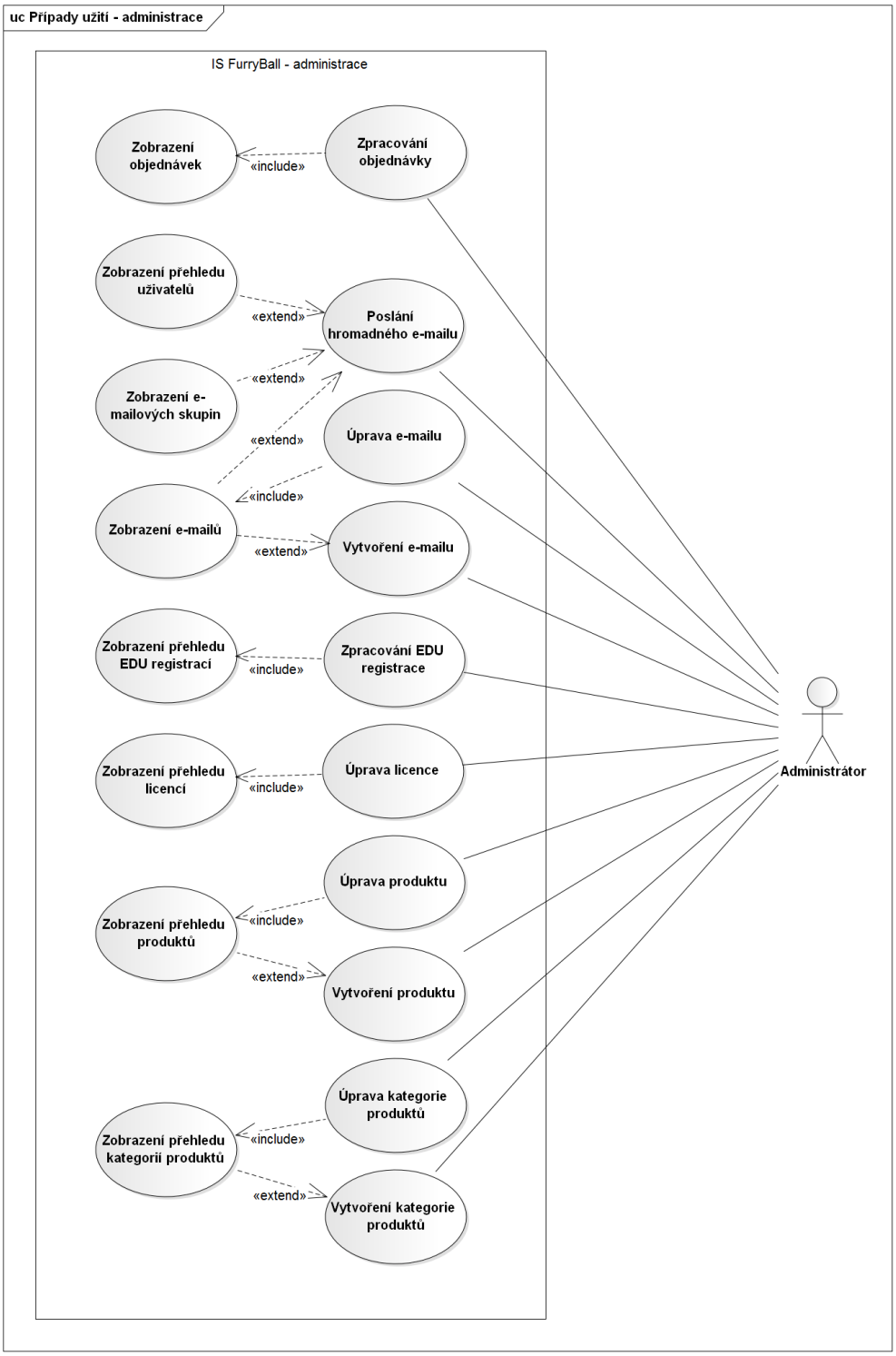
- údaje o uživateli
- přehled položek objednávky
- celkovou cenu
- stav objednávky (*čeká na zaplacení, zaplacená* apod.)

UC13. *Zpracování objednávky* – umožní administrátorovi zpracovat objednávku a vytvořit/upravit na jejím základě odpovídající licence.

1. Scénář začíná v okamžiku, kdy se administrátor rozhodne zpracovat objednávku.
2. administrátor zobrazí přehled objednávek – UC12.
3. Vybere jednu (nezpracovanou) z nich a zkontroluje její údaje.
4. Označí všechny nebo jen některé položky objednávky a zvolí možnost *'process'*.
5. Systém vybrané položky zpracuje, upraví na jejich základě odpovídající licence a informuje administrátora o výsledku.
6. Systém odešle objednateli e-mail informující ho o vyřízení objednávky.

UC14. *Zpracování EDU registrace* – umožní administrátorovi schválit EDU registraci.

1. Scénář začíná v okamžiku, kdy se administrátor rozhodne zpracovat EDU registraci.
2. Administrátor zobrazí seznam EDU registrací.
3. Vybere jednu (nezpracovanou) z nich a zkontroluje její údaje.
4. Pokud jsou údaje v pořádku, zvolí možnost *'zpracovat'*.
5. Systém schválí danou EDU registraci a informuje o výsledku administrátora.



Obrázek 2.3: Diagram případů užití - administrační rozhraní

6. Systém odešle uživateli žádajícímu o EDU registraci e-mail s informací o vyřízení registrace.

UC15. *Zobrazení přehledu uživatelů* – umožní administrátorovi zobrazit přehled všech uživatelů, kde u každého uživatele zobrazuje zejména:

- jeho kontaktní informace
- počet jeho licencí s odkazem na jejich přehled

UC16. *Filtrování uživatelů* – umožňuje administrátorovi v přehledu uživatelů jejich snadné filtrování, a to podle:

- uživatelského jména
- země
- počtu licencí
- aktivní/neaktivní (blokovaný)
- je/není plátce daně

UC17. *Odeslání hromadného e-mailu* – umožní administrátorovi odeslat e-mail na více adres najednou, a to buď jejich manuálním výběrem, nebo zvolením jedné nebo více e-mailových skupin (skupina e-mailů je samostatná entita).

1. Scénář začíná v okamžiku, kdy se administrátor rozhodne poslat hromadný e-mail. K samotnému formuláři pro odeslání se může dostat třemi způsoby:
   - Výběrem několika emailů z jejich přehledu
   - Výběrem několika e-mailových skupin z jejich přehledu
   - Výběrem několika uživatelů z jejich přehledu
2. Systém zobrazí formulář pro hromadný e-mail.
3. Administrátor vyplní předmět a tělo e-mailu, příp. přidá/odebere adresáty a formulář odešle.
4. Pokud je formulář vyplněn správně, systém e-mail odešle na dané adresy a informuje administrátora o výsledku.

UC18. *Vytvoření nového produktu* – umožní administrátorovi vytvořit nový produkt. Formulář pro vytvoření bude přístupný z přehledu produktů.

2.7 Doménový model

Na základě předchozí analýzy jsem vytvořil doménový model, viditelný na diagramu 2.4. V této kapitole popíšu nejdůležitější entity a jejich vztahy.

2.7.1 Entity

2.7.1.1 User

Entita User představuje uživatele systému. Ten je identifikován uživatelským jménem a e-mailem. Uživatelské jméno zároveň identifikuje uživatele na podpůrném fóru. Pro nákup licencí musí uživatel vyplnit i další údaje (jméno, adresa a další, viz 2.4).

2.7.1.2 Role

Role vyjadřuje práva uživatele v systému, např. role *administrátor* má přístup k administračnímu rozhraní, zatímco role *uživatel* ne. Zároveň role určuje (ve spojení s entitou ProductCategory), jaké produkty může daný uživatel koupit.

2.7.1.3 Licence

Licence umožňuje uživateli používat software FurryBall na jednom počítači. Je identifikována pomocí HWID (viz 2.7.1.6) a příslušností k SuperLicenci (2.7.1.4).

2.7.1.4 SuperLicence

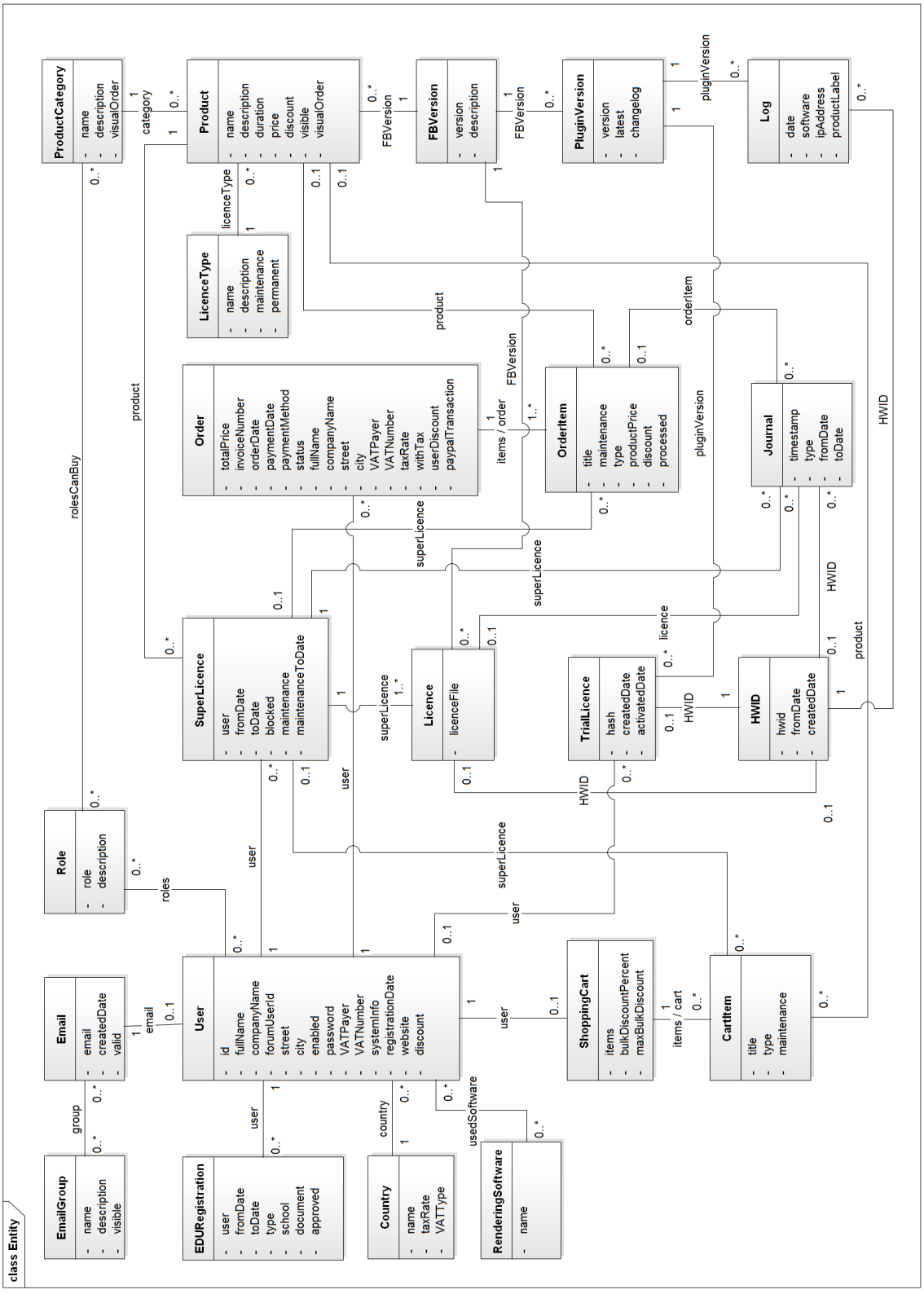
SuperLicence je entita, která sdružuje více licencí. Je potřeba z toho důvodu, že když je licence upgradována (ať už automaticky, nebo uživatelem), vznikne nová licence a původní uživateli zůstane (viz UC10.). Už ji ale není možno upgradovat znovu. Zároveň, pokud původní licence byla časově omezená, nová licence toto omezení sdílí. Efektivně jde tedy o skupinu licencí, které sdílejí určité vlastnosti a navzájem se ovlivňují. A tuto skupinu právě představuje SuperLicence. Rozhodl jsem se proto, že každá licence bude mít svou SuperLicenci (i když je ve skupině sama) a sdílené vlastnosti budou v SuperLicenci.

2.7.1.5 TrialLicence

Entita TrialLicence představuje zkušební verzi softwaru FurryBall, kterou si (neznámý) uživatel nainstaloval na svůj počítač. Zkušební verzi je třeba před použitím a poté jednou za čas aktivovat přes internet (jde o zcela jiný proces než případ užití UC9. – *Aktivace licence* a není součástí této bakalářské práce). Je třeba zmínit, že přes podobný název tvoří zkušební licence a standardní licence dva oddělené funkční celky.

2.7.1.6 HWID

HWID je jednoznačný identifikátor konkrétního počítače.



Obrázek 2.4: Doménový model

2.7.1.7 Product

Tato entita představuje produkty nabízené k prodeji (licence). Kromě vlastních atributů (název, cena apod.) má vazbu na ProductCategory (viz 2.7.1.8), LicenceType (typ licence, který vznikne koupením tohoto produktu) a FBVersion (major verze softwaru FurryBall, v rámci které jsou aktualizace zdarma).

2.7.1.8 ProductCategory

Kategorie produktů – určuje, kteří uživatelé mohou koupit který produkt na základě jejich rolí (a v současnosti určuje i členění na stránce s nabídkou produktů).

2.7.1.9 Order

Entita Order představuje objednávku produktů uživatelem. Má vazbu na uživatele a zároveň kopii všech jeho dat, která se objednávky týkají (jméno, adresa, zda je plátcem daně apod.). Dále obsahuje kolekci položek objednávky.

2.7.1.10 OrderItem

Položka objednávky (jeden produkt) – má vazbu na produkt a zároveň kopii jeho dat, která se jí týkají. Dále má vazbu na svou rodičovskou objednávku a volitelně na SuperLicenci, pokud tuto SuperLicenci upravuje (upgrade, prodloužení platnosti) místo aby vytvářela novou.

2.7.1.11 Journal

Entita Journal slouží k záznamu historie (super)licencí. Při každé důležité změně se vytvoří nový záznam obsahující typ změny a vazby na entity, které se změny účastní. Typy zaznamenávaných změn:

- nová superlicence
- obnovení platnosti dočasné superlicence
- upgrade superlicence objednaný uživatelem
- automatický upgrade superlicence na základě vydání nové verze
- aktivace licence
- změna HWID licence

Kapitola 3

Návrh

Tato kapitola popisuje proces návrhu systému. Cílem návrhu je podrobně specifikovat fungování systému vycházející z předchozí analýzy. Tento návrh už není obecný – je závislý na zvolených technologiích, jejichž popis je proto první částí kapitoly. V dalších částech se zabývám popisem architektury systému, uživatelského rozhraní a také databázového modelu.

3.1 Technologie

V této části se zabývám zvolenými technologiemi. Část z nich vychází z nefunkčních požadavků (kapitola 2.4.2), u ostatních vždy kromě popisu uvedu i zdůvodnění jejich výběru.

3.1.1 Back-end

3.1.1.1 LAMP stack

Jelikož je součástí nefunkčních požadavků i požadavek na konkrétní hosting a programovací jazyk, základní platforma je tímto určena. Požadovaný hosting *Savana 3000* poskytuje virtual private server (VPS) s operačním systémem Linux a webovým serverem Apache, zároveň je pro implementaci zvolen jazyk PHP (verze 5) a pro ukládání dat je požadována databáze MySQL. Ve výsledku jde tedy o tzv. *LAMP stack*[1].

3.1.1.2 Framework – Symfony

Při tvorbě webových (ale i jiných) aplikací je mnoho součástí systému, které se v jednotlivých projektech opakují – vždy je třeba namapovat URL na aplikační logiku, vyřešit zabezpečení, propojit jednotlivé vrstvy aplikace atd. V případě jazyka PHP vznikaly pro řešení těchto opakujících se úloh nejdříve samostatné knihovny, zajišťující např. pouze komunikaci s databází nebo složící jako šab-

lonovací systém. Později (s příchodem PHP 5 a jeho vylepšeným objektovým modelem) začaly vznikat celé frameworky, poskytující nejen řešení pro jednotlivé oblasti webové aplikace, ale spojující tato řešení do funkčního celku [2].

Symfony je jedním z PHP frameworků pro tvorbu webových aplikací. Mezi jeho základní charakteristiky patří objektový přístup, vysoká modularita a znovupoužitelnost komponent (části frameworku používá např. Drupal[11] nebo Joomla[12] [3]), používání konzolových příkazů pro usnadnění práce s frameworkem nebo využívaní dependency injection (viz 3.2.6). Výhodou Symfony je i velká komunita vývojářů a kvalitní dokumentace.

Z těchto důvodů byl zadavatelem zvolen framework Symfony jako nejvhodnější pro navrhovaný systém.

3.1.1.3 Persistence dat – Doctrine

Pro persistenci dat jsem zvolil knihovnu *Doctrine*, používající objektově relační mapování (ORM). ORM umožňuje mapovat objekty (datové třídy v objektově orientovaném programování) na relační databázi a odstiňuje programátora od způsobu, jakým jsou v databázi uloženy. Mapování probíhá pomocí konfiguračních souborů nebo pomocí anotací přímo ve zdrojovém kódu tříd. Výhodou ORM je větší přehlednost systému, není potřeba „přepínat“ mezi objektovým a relačním přístupem, a i počáteční rychlost vývoje je vyšší, protože základní operace s daty (tzv. CRUD[13] operace) není třeba definovat. [4] Mezi nevýhody patří přidání další abstraktní vrstvy a tím snížení výkonu aplikace (tomu se ale dá správným použitím do značné míry předejít) a také obtížnost převedení některých objektových principů do relačního paradigmatu a z toho vyplývající nebezpečí špatného návrhu databáze. [5]

Pro použití *Doctrine* jsem se rozhodl proto, že výhody podle mě převažují nad nevýhodami (resp. nevýhodám se lze do určité míry vyhnout) a použití knihovny je v ekosystému Symfony de facto standardem.

3.1.1.4 Správa uživatelů – FOSUserBundle

FOSUserBundle (slovem „bundle“ se v ekosystému Symfony označuje modul poskytující určitou funkčnost – v dalším textu znamenají slova *bundle* a *balíček* to samé) je balíček usnadňující správu uživatelů – poskytuje základní aplikační logiku pro persistenci uživatelů, jejich registraci, přihlašování, zobrazení a úpravu profilu a další funkce. Každá část funkčnosti je redefinovatelná. [6]

[11] open-source content management system (CMS), `https://www.drupal.org/`
[12] CMS, `http://www.joomla.org/`
[13] z anglického create-read-update-delete (CRUD)

Pro Symfony existuje mnoho balíčků pro práci s uživateli, ale *FOSUserBundle* je jednoznačně nejrozšířenější, je pravidelně aktualizován a má kvalitní dokumentaci. Proto jsem se rozhodl pro něj.

3.1.1.5 Administrační rozhraní – SonataAdminBundle

SonataAdminBundle slouží k jednoduché tvorbě administračních rozhraní webových aplikací. Je součástí *Sonata Project*, což je kolekce vzájemně spolupracujících balíčků poskytujících funkce jako CMS, search engine optimization (SEO) a právě administrace dat. [7] Podobně jako *FOSUserBundle* poskytuje základní funkčnost, kterou lze upravit a rozšiřovat.

Z funkčních požadavků na administraci (2.4.1.2) vyplývá, že značná část funkčnosti bude spočívat v základních CRUD operacích, pro které se standardizované řešení jako *SonataAdminBundle* přesně hodí. Zároveň jde o nejvíce používaný balíček pro daný účel.

3.1.1.6 E-commerce balíčky

Pro Symfony existuje několik e-commerce balíčků, mezi hlavními *Sylius*[14], *Thelia 2*[15] a *Elcodi*[16]. Jde ve všech případech o komplexní řešení pro celý proces nákupu, od organizace a prezentace produktů, přes samotný nákup až po doručování objednávek. Jejich nevýhodou je pro navrhovaný systém zbytečná komplexnost a také nekvalitní (v případě *Elcodi* takřka neexistující) dokumentace. Proto jsem se rozhodl vytvořit vlastní řešení.

3.1.1.7 Šablonovací systém – Twig

Šablonovací systémy slouží k oddělení prezentace dat od aplikační logiky. To zvyšuje znovupoužitelnost a usnadňuje spolupráci back-end a front-end vývojářů. Zároveň tyto systémy poskytují i další výhody jako např. automatické "escapování"proměnných[17].

Z mnoha pro PHP dostupných systémů (např. *Smarty*, *Blade* nebo české *Latte*) jsem zvolil *Twig* – kromě výše popsaného obsahuje i dědičnost šablon, která umožňuje další redukci množství kódu[9] a hlavně je zároveň součástí standardní distribuce Symfony.

[14] `http://sylius.org/`

[15] `http://thelia.net/`

[16] `http://elcodi.io/`

[17] ošetření výstupu tak, aby nemohlo dojít k žádným nevyžádaným vedlejším efektům, např. spuštění skriptu[8]

3.1.1.8 Internacionalizace – Translator

Jako internacionalizace se označuje proces, kdy se aplikace vytváří tak, že je možno ji beze změn v kódu používat v různých jazykových mutacích. V Symfony tuto funkčnost zajišťuje komponenta *Translator*, která funguje na principu abstrahování přeložitelného obsahu do párů *klíč: hodnota*, kdy v kódu aplikace se použije klíč a ten se při prezentaci dat nahradí hodnotou podle jazyka uživatele.[10]

3.1.1.9 Správa statických zdrojů – Assetic

Pro správu statických zdrojů (cascading style sheet (CSS) soubory, Javascript (JS) soubory) používám knihovnu *Assetic* (resp. balíček *AsseticBundle*). Je totiž provázaný se Symfony, a na rozdíl od samostatných řešení jako *Grunt*[18] nebo *Gulp*[19] si vystačí s PHP (obě uvedené alternativy potřebují na serveru Node.js[20].

3.1.1.10 Napojení na podpůrné fórum

Vedle vlastního webu FurryBall (`furryball.aaa-studio.eu`) běží na serveru také fórum pro zákaznickou podporu. Použitým řešením je *phpBB*[21], což je fórum napsané v PHP a využívající některé komponenty Symfony (ale ne framework jako takový).[11]

Funkční požadavek FU2. specifikuje, že je třeba při registraci vytvořit automaticky i účet na fóru. Jelikož se fórum nachází na stejném serveru, je možné volat jeho funkce přímo. Fórum obsahuje i hledanou funkci `user_add`. *phpBB* není psáno v objektovém stylu – používá globální funkce i globální proměnné, jejichž použití je ve funkcích třeba deklarovat (klíčovým slovem `global`). Toto je ale omezeno jen na konkrétní třídu, která má spojení s fórem na starosti.

3.1.2 Front-end

3.1.2.1 Javascript a jQuery

Javascript je skriptovací jazyk používaný hlavně pro webové stránky a vykonávaný ve webovém prohlížeči, tedy na straně klienta (v současné době se ale používá i jinde, např. v *Node.js* nebo v databázi *CouchDB*[22]).[12]

[18] `http://gruntjs.com/`

[19] `http://gulpjs.com/`

[20] serverový framework pro webové aplikace psané i na serveru v Javascriptu, `https://nodejs.org/`

[21] `http://www.phpbb.cz/`

[22] `http://couchdb.apache.org/`

jQuery je javascriptová knihovna, která (mimo jiné) usnadňuje procházení a manipulaci s Document Object Model (DOM), obsluhu událostí, základní práci s animacemi a Ajax[23] požadavky.[13]

Nefunkční požadavek N4. specifikuje, že výsledná webová aplikace bude součástí existujícího webu. Ten používá javascript jen omezeně (hlavně na kontrolu/přidání/odebrání prvku formulářů), proto i uživatelské rozhraní navrhovaného systému bude převážně statické a javascript bude používat pro podobné účely, aby nedocházelo k nekonzistenci v uživatelském prožitku.

3.1.2.2 Twitter Bootstrap

Twitter Bootstrap je front-endový framework pro webové aplikace, který zrychluje realizaci uživatelského rozhraní. Poskytuje sadu předpřipravených CSS pravidel a dalších prvků a umožňuje k HTML prvkům přistupovat jako ke komponentám uživatelského rozhraní.[14]

V navrhovaném systému používám hlavně části frameworku zajišťující rozvržení na stránce a jednoduché interakce pomocí javascriptu. Části určující vizuální podobu jednotlivých prvků naopak využívám minimálně, protože jsou často v konfliktu s existující podobou webu.

3.2 Architektura a principy frameworku Symfony

V této kapitole popíšu základní architekturu a principy frameworku Symfony.

3.2.1 Request/Response

PHP frameworky se často popisují jako Model View Controller (MVC)[24] frameworky. Symfony se takto neprezentuje (aspoň od verze 2), a spíše sebe sama popisuje jako *HTTP framework* nebo *Request/Response framework*.

„Potřebujete něco, co zpracuje Požadavek a vrátí Odpověď“[15]

Princip Symfony je takto jednoduchý – na základě příchozího požadavku se vytvoří objekt typu Request a na konci se vrátí objekt typu Response. Programátor má na starosti právě oblast mezi těmito dvěma body.

3.2.2 Front Controller

Návrhový vzor *Front Controller* označuje u webových aplikací jednotný vstupní bod pro všechny příchozí požadavky.[16]. Symfony má v základu 2 front cont-

[23] asynchronní požadavky pomocí javascriptu
[24] návrhový vzor pro architekturu aplikace

rollery – `app.php` a `app_dev.php`. Oba se nacházejí v kořenovém adresáři webu a první z nich je určen pro *produkční prostředí* a druhý pro *vývojové prostředí*. Rozdíl mezi nimi je ten, že `app_dev.php` přidává do aplikace prvky pro ladění a diagnostiku (*Symfony profiler*[17]) a také hlídá změny v celém kódu aplikace (a je kvůli tomu výrazně pomalejší). Front controller pro produkční prostředí naproti tomu používá cache zdrojových souborů.

3.2.3 Routování

Proces routování znamená namapování URL na konkrétní akci aplikace, tedy že např. adresa *www.website.com/homepage* zobrazí domovskou stránku aplikace. V Symfony se toto nastavuje konfigurací, a to buď v konfiguračních YAML/XML souborech, nebo anotacemi přímo ve zdrojovém kódu (tuto možnost jsem zvolil já).

3.2.4 Controllery

Controller je třída, která obsahuje metody pro obsluhu HTTP požadavků[25]. Těmto metodám se říká *akce*. Jednoduchý controller může vypadat např. takto:

```
namespace AppBundle\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class ExampleController extends Controller
{
    /**
    * @Route("/hello/{name}", name="hello")
    */
    public function helloAction($name)
    {
        return new Response('<html><body>Hello '.$name.'!</body></html>');
    }
}
```

Metoda `helloAction` je namapována pomocí anotace `@Route` na adresu `/hello/{name}` pod jménem „hello“ – toto jméno se pak používá pro odkazování na tuto adresu. Část ve složených závorkách (`{name}`) je zástupný řetězec, na jehož místě

[25]Dokumentace Symfony je v tomto ohledu nejednoznačná – někde se jako controller označuje celá třída, jinde každá jednotlivá metoda. Já budu pod pojmem controller vždy uvažovat celou třídu.

může být cokoliv a tato hodnota se pak vloží do parametru `$name` metody `helloAction`. Uvnitř této metody se mohou provést libovolné příkazy, ale na konci musí metoda vrátit objekt typu `Response`.

3.2.5 Služby

Teoreticky by všechna logika mohla být v controllerech, ale to by vedlo k nepřehlednému kódu a často i k jeho duplikaci. Controller by měl obsahovat jen malé množství kódu potřebného pro spojení různých částí systému.[18] Jedním ze způsobů, jak toho dosáhnout, je použití tzv. služeb (services), což jsou třídy obsahující nějakou jasně definovanou aplikační logiku (často je tato logika vázána na konkrétní entity, takže vznikne např. `UserService`, obsahující logiku pro práci s uživateli).

Pro přístup k těmto službám zavádí Symfony tzv. *Service container*, což je kontejner obsahující všechny služby, uložené jako klíč – hodnota. Druhým způsobem jak přistupovat ke službám je jejich *injektování* do jiných služeb.

3.2.6 Dependency injection

Služby se v kontejneru neocitnou jen tak samy od sebe – je třeba nějak definovat, jaké třídy a s jakými parametry se mají vytvořit a do kontejneru vložit. Symfony pro toto používá *dependecy injection* (česky někdy „vkládání závislostí“), což je technika umožňující definování závislostí mezi třídami a jinými částmi programu tak, že třída sama nemá zodpovědnost za získávání objektů, které potřebuje ke své činnosti.[19] Místo toho se jí objekty předají při jejím vytváření (jako parametry konstruktoru), nebo někdy později (pomocí setteru[26]).[20] V Symfony (a PHP obecně) se používá hlavně první způsob (kostruktor).

Služby se definují v konfiguračním souboru ve formátu YAML nebo XML např. takto:

```
// parameters.yml

licence_service:
    class: Simon\FurryBallBundle\Service\LicenceService
    arguments:
        - '@doctrine'
        - '@licence_generator'
        - '@journal_service'
```

[26] metoda přiřazující hodnotu členské proměnné

Tedy definuje se jméno služby (1. řádek), její třída a parametry konstruktoru (`@` na začátku parametrů označuje, že vkládáme jinou službu).

K samotnému kontejneru se přistupuje v controllerech takto:

```
// ExampleController.php
...
$this->get('licence_service')->doSomething();
...
```

Controller je také jediné místo, kde bychom měli přistupovat k celému kontejneru, do služeb a jiných objektů injektujeme zásadně jen ty služby, které daný objekt opravdu potřebuje [21].

3.2.7 Šablony

Jak jsem psal v kapitole 3.1.1.7, používám šablonovací systém Twig. Ten umožňuje oddělit prezentaci dat od ostatní logiky a jeho použití v rámci Symfony je jednoduché – místo, abychom v controlleru přímo vytvářeli instanci `Response`, zavoláme metodu `render()`:

```
// ExampleController.php
...
public function helloAction($name)
{
// misto
// return new Response('<html><body>Hello '.$name.'!/body></html>');

// zavolame
    return $this->render('template.html.twig, array('name' => $name));
}
...
```

Druhý parametr metody `render()` je pole parametrů, které se mají předat šabloně.

3.2.8 Konvence

Symfony umožňuje dodržováním určitých konvencí zkrátit a zjednodušit kód aplikace (princip *convention over configuration*). Tyto konvence není nutné dodržovat, ale je pak potřeba některé věci nakonfigurovat ručně. Dodržování konvencí na druhou stranu má i nevýhodu – může (paradoxně) učinit kód nepřehledným, pokud ten, kdo ho zkoumá, konvence nezná.

Já jsem se rozhodl konvence dodržovat, protože se domnívám, že zjednodušení kódu a zrychlení vývoje více než vyvažuje zmíněnou nevýhodu. Konkrétní využití konvencí popíšu v dalších kapitolách.

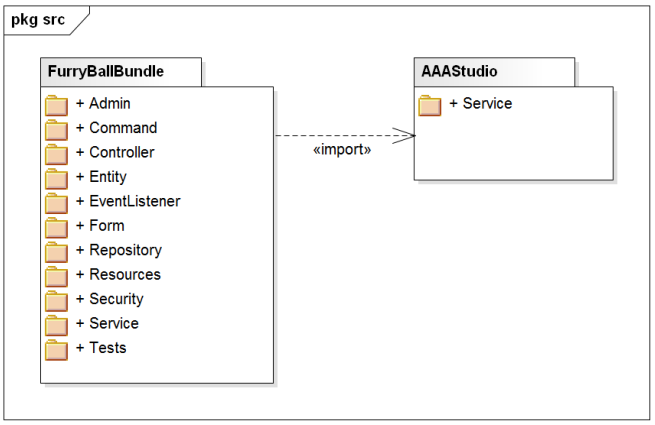
3.2.9 Adresářová struktura

Doporučená adresářová struktura Symfony aplikace vypadá takto:

- `app` ..... základní složka aplikace
  - `cache` ..... dočasné soubory aplikace
  - `config` ..... složka s hlavními konfiguračními soubory
  - `logs` ..... logy
  - `appKernel.php` ..... třída načítající konfiguraci a použité balíčky
- `src` ..... zdrojové kódy projektu
- `vendor` ..... použité knihovny třetích stran
- `web` ..... kořenový adresář webu
  - `app.php` ..... front controller pro produkční prostředí
  - `app_dev.php` ..... front controller pro vývojové prostředí

3.3 Architektura navrhovaného systému

V této kapitole ukážu, jak navrhovaný systém využívá a rozšiřuje architekturu danou zvoleným frameworkem. Zároveň popíšu některé prvky architektury, které nejsou přímo nedílnou součástí frameworku a nezmiňoval jsem je tedy v předchozí kapitole.



Obrázek 3.1: Základní balíčky

Systém dodržuje základní doporučenou strukturu z minulé kapitoly. Všechny zdrojové kódy aplikace budou tedy ve složce `/src` (až na několik výjimek, kdy

Obrázek 3.2: FurryBallBundle – struktura

jinou organizaci vyžadují knihovny třetích stran). Aplikace je rozdělena do dvou základních balíčků (jak je vidět na obrázku 3.1):

- *FurryBallBundle* – hlavní funkčnost aplikace
- *AAAStudio* – služby potřebné pro napojení na existující web a další části starého systému

3.3.1 Balíček FurryBallBundle

FurryBallBundle je hlavním balíčkem celé aplikace a obsahuje většinu funkčnosti – má na starosti uživatelský účet a administrační rozhraní systému. Jeho základní struktura je na diagramu 3.2 a dodržuje konvence pro Symfony aplikace.

3.3.1.1 Controller

Složka `Controller` obsahuje všechny controllery systému. Ty jsou vstupními body aplikace a všechny požadavky jdou přes ně. Složka je dále členěna takto:

- `AAAController` – obsahuje controller pro stávající web
- `Admin` – obsahuje všechny controllery administračního rozhraní
- `UserAccount` – obsahuje všechny controllery pro uživatelský účet na webu

- `FBClient` – obsahuje controller pro přijímání požadavků z desktopových klientů FurryBall

3.3.1.2 Service

Obsahuje všechny služby (viz 3.2.5) balíčku. Jde většinou o služby určené pro práci s konkrétní entitou, např. `LicenceService`, `OrderService` nebo `EmailService`. Dále jsou zde i služby poskytující funkčnost jiným službám, např. `PayPalService` (mající na starosti komunikaci s platebním systémem PayPal) a `MailerService` (služba pro posílání e-mailů).

Zvláštní kategorií jsou tzv. „dummy“ služby, které slouží pro vývoj a testování na localhostu. Např. `DummyForumService` simuluje službu `ForumService`, protože na localhostu není fórum přístupné.

3.3.1.3 Entity

Složka `Entity` obsahuje (překvapivě) všechny entity systému. Tyto entity vycházejí z doménového modelu (kapitola 2.7) a většinou jsou ukládány do databáze. Mapování entit na databázi zajišťuje *Doctrine* (viz 3.1.1.3) pomocí anotací.

3.3.1.4 Repository

Tato složka obsahuje tzv. *repository třídy*. Tyto třídy slouží k definování složitějších databázových dotazů, které se zabalí do metod této třídy a jsou pak k dispozici v celém systému. Každá *Doctrine* entita má automaticky repository třídu (jde o třídu `Doctrine\ORM\EntityRepository`) obsahující základní dotazy (`find(id)`, `findAll()`, ...), ale tuto třídu je možné nahradit svou vlastní, dědící od `EntityRepository`:

```
class ProductRepository extends EntityRepository
{
    public function findAllOrderedByName()
    {
        return $this->getEntityManager()
            ->createQuery(
            'SELECT p FROM AppBundle:Product p ORDER BY p.name ASC'
        )
        ->getResult();
    }
}
```

(příklad je převzat z dokumentace Symfony [22])

Použitým jazykem pro databázové dotazy je *Doctrine Query Language (DQL)*. Ten je velmi podobný SQL, ale místo s řádky tabulek pracuje s objekty.

3.3.1.5 Form

Ve složce `Form` se nacházejí všechny formuláře použité v systému. Jelikož je Symfony objektově orientovaný framework, i formuláře jsou v něm objekty. Formuláře lze definovat v controllerech, ale pokud chceme nějaký formulář použít na několika místech, je lepší vytvořit *formulářovou třídu* (form class), ve které definujeme vlastnosti formuláře a pak ho můžeme použít kdekoliv v aplikaci. Takto například vypadá formulář pro editaci e-mailu:

```
class EmailType extends AbstractType
{
    /**
    * @param FormBuilderInterface $builder
    * @param array $options
    */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('email', null, array(
            'label' => 'form.user.email',
            ));
    }

    /**
    * @param OptionsResolverInterface $resolver
    */
    public function setDefaultOptions(OptionsResolverInterface $resolver)
    {
        $resolver->setDefaults(array(
        'data_class' => 'Simon\FurryBallBundle\Entity\Email'
        ));
    }

    /**
    * @return string
    */
    public function getName()
    {
        return 'form_email';
    }
}
```

3.3.1.6 Resources

Složka `Resources` obsahuje podpůrné soubory (obecně vše, co není PHP třída).

- `config` – konfigurační YAML soubor balíčku (definice služeb)
- `public` – veřejné zdroje balíčku – JS, CSS soubory
- `translations` – YAML soubory pro lokalizaci (viz 3.1.1.8).
- `views` – všechny šablony balíčku

3.3.1.7 Security

Tato složka obsahuje třídy související se zabezpečením.

3.3.1.8 Admin

Ve složce `Admin` se nacházejí třídy definující administrační rozhraní systému. Pro jeho tvorbu je použit *SonataAdminBundle* (viz 3.1.1.5). Každá třída dědí od `Sonata\AdminBundle\Admin\Admin` je vázaná na konkrétní entitu (např. `EmailAdmin` na entitu `Email`) a určuje, jak se budou data dané entity v administračním rozhraní zobrazovat a filtrovat a jak s nimi půjde manipulovat.

3.3.1.9 Tests

Ve složce `Tests` se nacházejí funkční testy systému.

3.3.2 Balíček AAAStudio

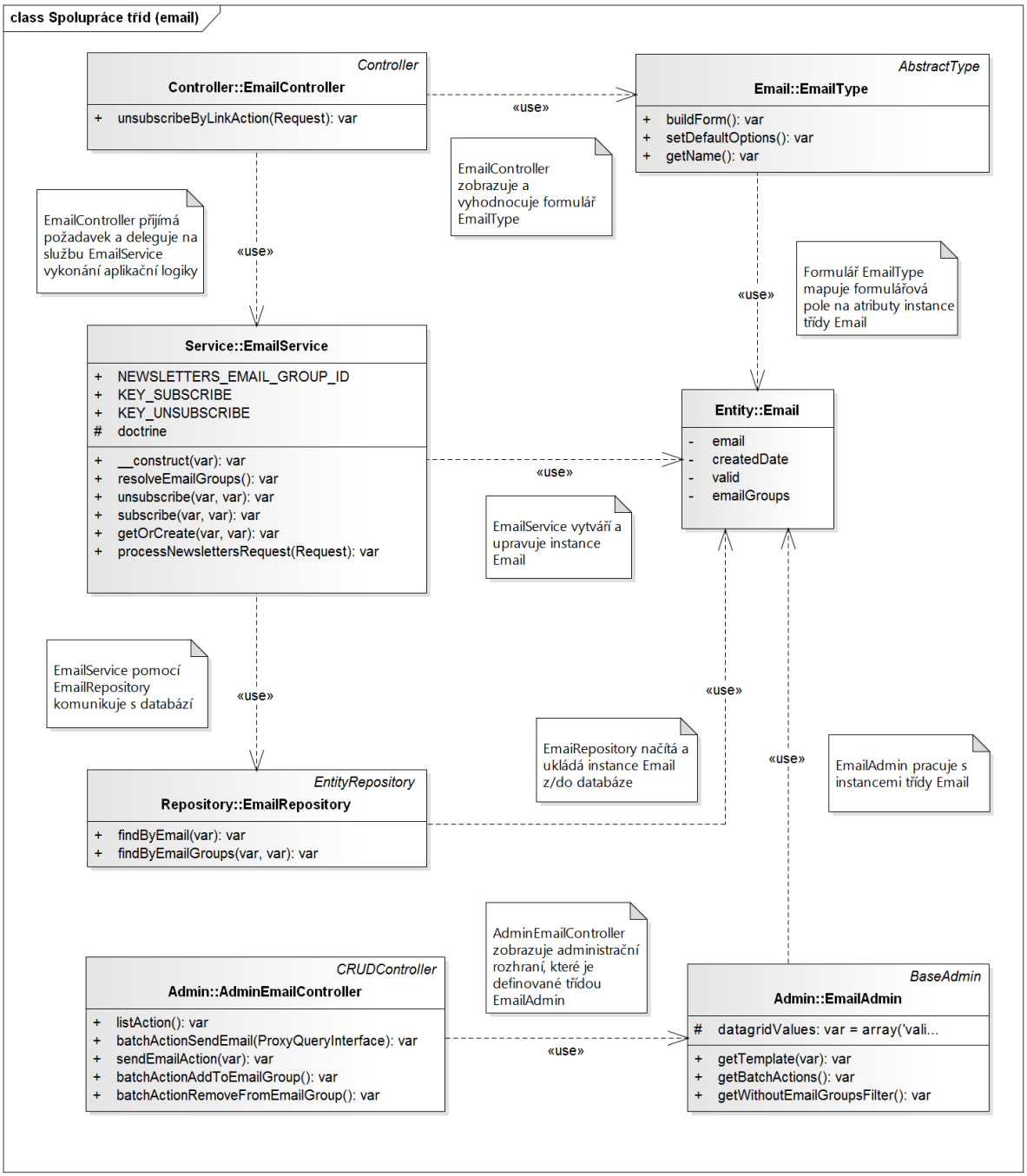
Balíček `AAAStudio` obsahuje všechnu funkčnost, která vychází ze starého informačního systému a kterou zařizuje zadavatel. Jde hlavně o načítání stránek původního webu (viz 3.3.4), knihovny pro napojení na platební brány (jejich použití je jedním z nefunkčních požadavků) a vytváření licenčních souborů. Spojení s novým systémem je zajištěno pomocí definovaných rozhraní.

3.3.3 Spolupráce tříd

Jednotlivé typy tříd (služby, controllery, formuláře, ...) tvoří vrstvy aplikace, které spolupracují na zpracování příchozího požadavku. Základním kamenem systému jsou entitní třídy, proto jsou controllery, služby a další třídy často navázané na konkrétní entitu – např. k entitě `Email` existuje `EmailController`, `EmailService` atp.

Na obrázku 3.3 je právě na příkladu entity `Email` ukázána spolupráce jednotlivých tříd s vysvětlením jejich vazeb.

Vazby na obrázku neplatí vždy – k některým entitám neexistuje např. controller nebo služba. Naopak některé služby nemají přidruženou entitu (existuje



Obrázek 3.3: Spolupráce tříd (vrstvy aplikace)

služba `PayPalService`, ale nikoliv entita `PayPal`). Jde spíše o přístup, který se aplikace snaží dodržovat.

3.3.4 Integrace stávajícího webu

Jedním z nefunkčních požadavků je i integrace do existujícího webu zadavatele. Jde o web psaný v čistém PHP (bez frameworku), jež používá javascriptový WISIWIG[27] editor pro tvorbu statických stránek, jejichž HTML kód je poté ukládán do databáze.

Jelikož do tohoto ad-hoc vytvářeného webu by bylo velmi složité integrovat navrhovaný systém, po prozkoumání jeho fungování jsem se rozhodl naopak starý web integrovat do nového systému. Jelikož jde o dva naprosto rozdílné systémy, toto spojení se asi nedá nazvat „čistým“, ale zadavatel si zcela nový web nepřeje a bylo by to i mimo rozsah této práce.

Spojení bude fungovat tak, že pokud URL příchozího požadavku není namapována na žádný controller nového systému, předá se tato URL konkrétní službě, která bude mít na starosti na základě tohoto požadavku načíst obsah stránky z databáze. Systém jako celek pak vrátí tuto stránku (více k tomuto tématu v kapitole 4 – Realizace).

3.4 Databázový model

3.4.1 Použití Doctrine

Aplikace používá pro persistenci dat ORM framework *Doctrine*. To znamená mimo jiné i to, že struktura databáze je definována anotacemi u samotných entit, které budou v databázi uloženy. Tyto anotace vyjadřují vztahy mezi entitami v objektovém paradigmatu[28] a *Doctrine* následně tyto vztahy převede (namapuje) na vztahy v relační databázi.

V návrhu je například entita User (reprezentující uživatele), která může mít jednu i více rolí (entita Role). Vztah mezi entitami v jejich kódu vypadá takto:

```
/**
* @ORM\Entity()
* @ORM\Table(name="fb_user")
*/
class User extends BaseUser
```

[27] What You See Is What You Get

[28] Toto platí jen přibližně – pokud je třeba využít konkrétní vlastnosti relační databáze, tato vlastnost se v anotacích také objeví (takže nejsme v čistě objektovém přístupu)

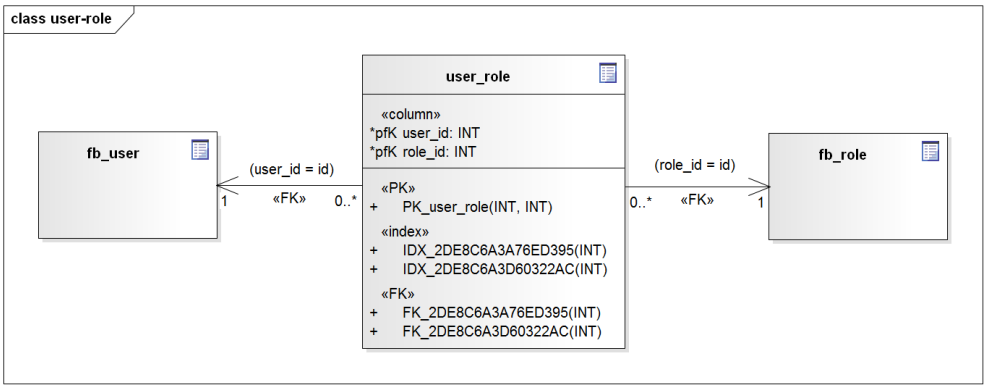
```
{
    ...

    /**
    * @var Role[]
    * @ORM\ManyToMany(targetEntity="Role") // definice vztahu m:n
    */
    protected $roles;

    ...
}

/**
* @ORM\Entity()
* @ORM\Table(name="fb_role")
*/
class Role implements RoleInterface
{
    ...
}
```

Doctrine takto definovaný vztah správně převede na vazební tabulku `user_role`, jak je vidět na obrázku 3.4.

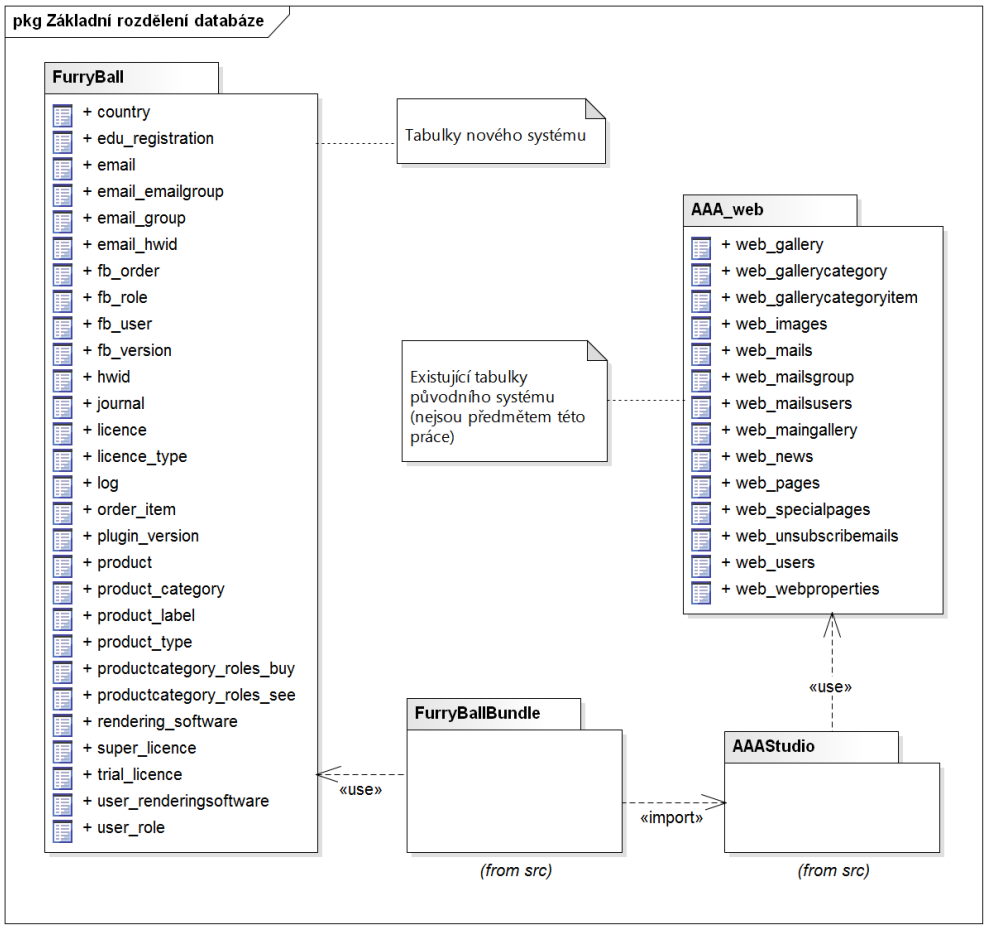


Obrázek 3.4: Relační datový model – základní rozdělení

3.4.2 Základní rozdělení databáze

Existující web FurryBallu funguje na principu načítání hotových stránek z databáze (viz 3.3.4). Jelikož je tento web integrován do nového systému, nacházejí

se v databázi i jeho tabulky. Nový systém ale tabulky používá pouze zprostředkovaně přes služby balíčku `AAAStudio` (hlavně služba `WebService`). Databáze se tak dá rozdělit do dvou nezávislých celků, kde každý z nich je využíván svým balíčkem (jak to ukazuje obrázek 3.5).



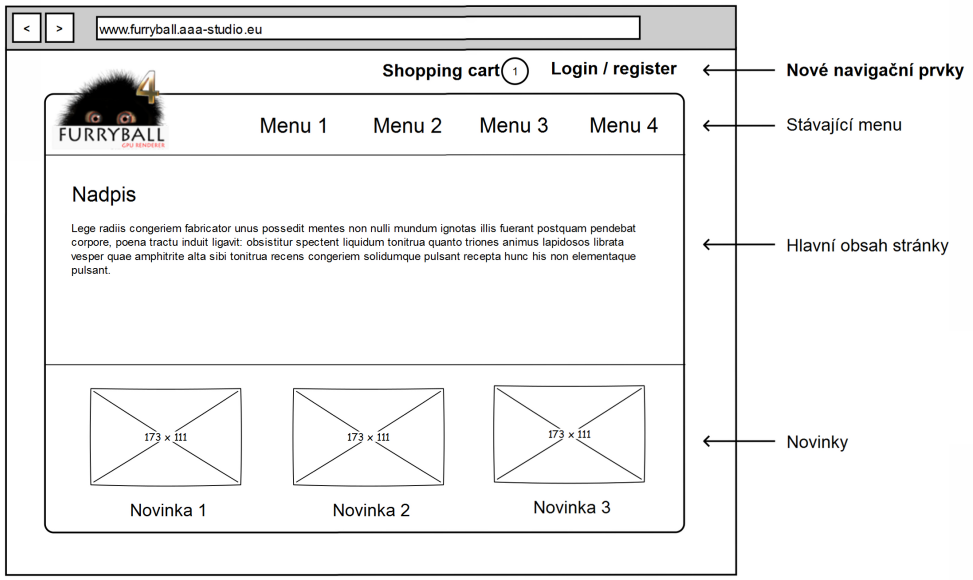
Obrázek 3.5: Relační datový model – základní rozdělení

Tabulky nového systému vycházejí z entit a jejich vztahů definovaných ve `FurryBallBundle/Entity`. Přehled všech tabulek se nachází v příloze B.

3.5 Uživatelské rozhraní

3.5.1 Uživatelský účet

Návrh uživatelského rozhraní klientské sekce je do značné míry ovlivněn požadavkem na jeho integraci do existujícího webu. Ten je převážně statický, resp. jde o kompletní stránky uložené v databázi a načítané PHP skriptem. I nový



Obrázek 3.6: Základní rozvržení – veřejný web

systém by proto bude načítat kompletní stránky a použití Javascriptu se omezí na pomocné funkce (např. skrytí formulářových polí) a drobné vizuální prvky.

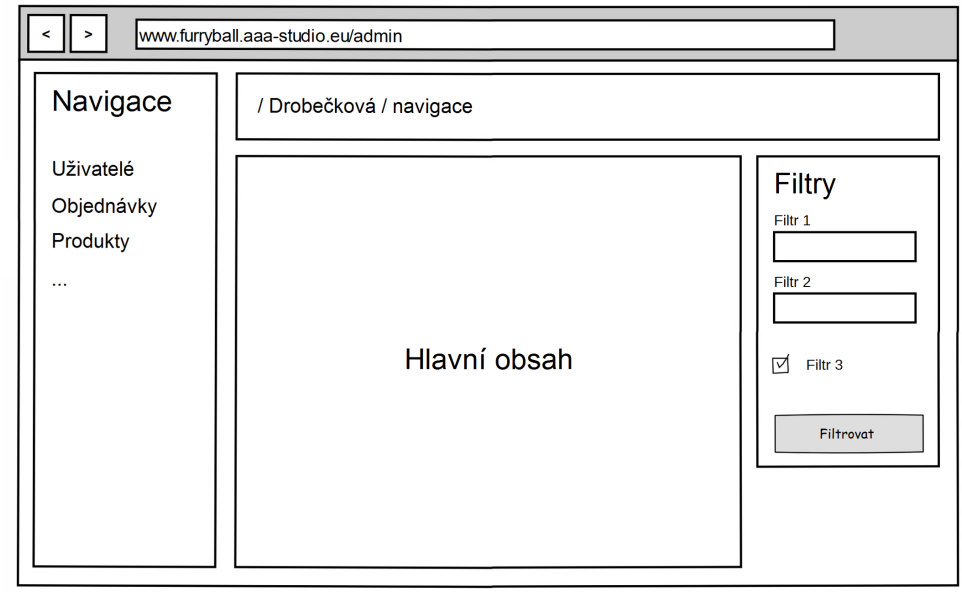
Rozvržení stávajícího webu používá klasické dělení header-content-footer, tedy hlavička s menu nahoře, pod ní hlavní obsah stránky a nakonec patička s doplňujícími informacemi, v tomto případě s novinkami (nové verze softwaru, speciální nabídky apod.). Do tohoto základního rozvržení je třeba umístit nové navigační prvky uživatelského účtu (přihlášení, registrace, nákupní košík, ...). Na základě konzultací se zadavatelem jsem se rozhodl umístit tyto prvky nad stávající menu mimo hlavní vizuální rámec stránky. Domnívám se totiž, že hlavním obsahem je stále existující web, a informace o přihlášení/odhlášení nebo o počtu položek v nákupním košíku je informací doplňkovou. Základní rozvržení s novými prvky ukazuje diagram 3.6.

Po přihlášení pak nové menu obsahuje odkazy na uživatelovy licence, objednávky a na jeho profil. Detailní podoba jednotlivých stránek je vidět v uživatelské příručce (příloha D.1).

3.5.2 Administrační rozhraní

Podoba administračního rozhraní je do značné míry dána použitým balíčkem (*Sonata Admin*). Základní rozvržení zobrazuje diagram 3.7. Toto rozvržení je možné změnit nahrazením standardních šablon vlastními [23], ale pro účely navrhovaného systému se ukázalo rozvržení jako plně dostačující a pouze u ne-

standardních obrazovek (poslání hromadného e-mailu) bude třeba vytvořit vlastní šablonu.



Obrázek 3.7: Základní rozvržení – administrace

Kapitola 4

Implementace

Tato kapitola pojednává o realizaci navrženého systému na základě návrhu z předchozí kapitoly. Uvedu zde způsob nasazení systému a také několik příkladů z jeho implementace, které považuji za zajímavé.

4.1 Adresářová struktura

Adresářová struktura aplikace dodržuje konvence Symfony a vypadá takto:

- `app` ..... základní složka aplikace
  - `cache` ..... dočasné soubory aplikace
  - `config` ..... složka s hlavními konfiguračními soubory
  - `logs` ..... logy
  - `Resources` ..... šablony pro balíčky třetích stran (viz dále)
  - `appKernel.php` ..... třída načítající konfiguraci a použité balíčky
- `src` ..... zdrojové kódy projektu
  - `AAAStudio` ..... balíček poskytující část funkčnosti starého systému
  - `Simon` ..... hlavní (mnou implementované) balíčky aplikace
    - `FurryBallBundle` ..... hlavní balíček aplikace
    - `UserBundle` ..... balíček rozšiřující FOSUserBundle (viz dále)
- `vendor` ..... použité knihovny třetích stran
- `web` ..... kořenový adresář webu
  - `bundles` ..... statické zdroje balíčků třetích stran
  - `css` ..... CSS soubory aplikace
  - `fonts` ..... písma použitá v aplikaci
  - `js` ..... JS soubory aplikace
  - `app.php` ..... front controller pro produkční prostředí
  - `app_dev.php` ..... front controller pro vývojové prostředí

V adresářové struktuře jsou dvě složky, které stojí za vysvětlení – jde o složky `/app/Resources` a `/src/Simon/UserBundle`. Symfony poskytuje mechanismus pro nahrazení některých částí balíčků třetích stran (hlavně šablon) pomocí umístění stejně pojmenovaných souborů na konkrétní místo v adresářové struktuře. Složka `/app/Resources` slouží právě k tomuto účelu. V tomto konkrétním případě se ve složce `/app/Resources/FOSUserBundle/views` nachází šablony nahrazující ty v původním `FOSUserBundle` (viz 3.1.1.4).

Podobně složka `/src/Simon/UserBundle` obsahuje PHP třídy nahrazující ty v `FOSUserBundle` (v terminologii Symfony se to označuje jako „dědičnost" balíčků [24]).

Podrobný popis tříd a jejich metod je v dokumentaci ve složce `/docs` na přiloženém CD.

4.2 Diagram nasazení

Diagram 4.1 zobrazuje nasazení systému. Systém je nasazen na hostingu *Savana 3000* (nefunkční požadavek N6.), tzn. webový server *Apache* ve verzi 2.4 a databáze *MySQL* verze 5.6. Se serverem komunikují jednak uživatelé (pomocí internetového prohlížeče) a také desktopoví klienti FurryBall (hlášení stavu, obnovování licence).

Podrobný popis nasazení systému najdete v příloze C – instalační příručka.

4.3 Vybrané příklady z implementace

4.3.1 Integrace stávajícího webu

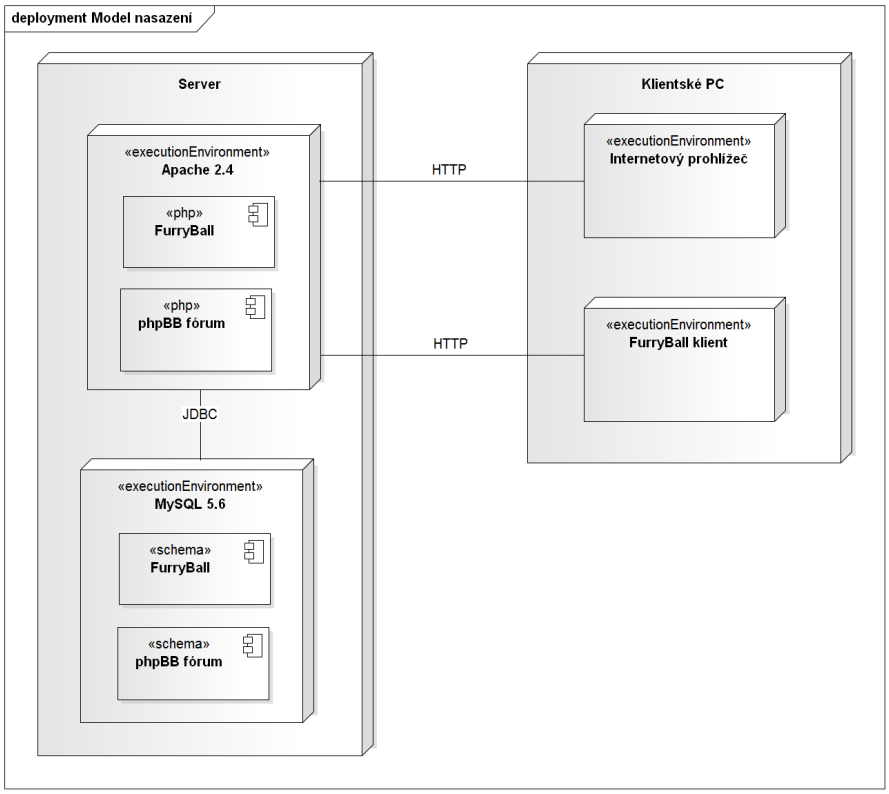
V kapitole 3.3.4 jsem popsal princip integrace stávajícího webu do nové aplikace. V této kapitole se chci tomuto tématu věnovat podrobněji.

Celý proces se skládá ze tří fází – nejdřív je třeba správně namapovat příchozí požadavek (rozpoznat, že jde požadavek na původní web), poté je potřeba načíst stránku z databáze, a nakonec stránku zobrazit.

4.3.1.1 Mapování požadavku

Rozpoznání požadavku na původní web je jednoduché díky tomu, že všechny jeho adresy mají tvar `zakladni-url/{prvni-cast}/{druha-cast}` (např. `zakladni-url/gallery/index.html`). Takže zpracování libovolného příchozího požadavku pak vypadá takto:

1. Hledám adresu v novém systému.



Obrázek 4.1: Diagram nasazení

2. Pokud jsem ji nenašel a pokud má požadovaná adresa tvar `/{prvni-cast}/{druha-cast}`, zkusím nalézt stránku na původním webu.
3. Pokud ani zde není, zobrazím 404 (not found) stránku.

Základním bodem pro routování (viz 3.2.3) v Symfony je soubor `app/config/routing.yml`. V tomto souboru může být buď přímo mapování adres na controllery, nebo (častěji) je zde specifikováno, kde (v jakých souborech) se mapování nachází. Důležité je pořadí, v jakém se adresy uvedou – v tomto pořadí pak budou kontrolovány pro každý příchozí požadavek a první odpovídající mapování se použije. Pro navrhovaný systém jsem tedy v `routing.yml` uvedl nejdřív adresy nové části a až po nich mapování pro stávající web:

```
# routing.yml
...
simon_furry_ball:  # mapování pro uživatelský účet
    resource: "@SimonFurryBallBundle/Controller/UserAccount"
                  # složka se zdroj. kódy
```

```
    type:     annotation  # mapování pomocí anotací ve zdroj. kódu
    prefix:   /

admin:  # mapování pro administrační rozhraní
    resource: '@SonataAdminBundle/Resources/config/routing/sonata_admin.xml'
    prefix: /admin
...
aaa_furryball_website:  # mapování pro existující web
    resource: "@SimonFurryBallBundle/Controller/AAAController"
    type:     annotation
    prefix:   /
```

4.3.1.2 Načtení a zobrazení stránky

O načtení stránky z databáze se stará třída `WebService` z balíčku `AAAStudio`. Tato třída implementuje `WebServiceInterface`, který definuje všechny metody potřebné k načtení stránky. Samotnou implementaci obstaral zadavatel.

Metody třídy `WebService` jsou volány z šablon. Diagram 4.2 ukazuje 3 základní případy, kdy se třídá `WebService` používá:

- *Každá stránka* – Na každé stránce se načítá horní menu webu a novinky na spodním okraji stránky. Např.:

  ```
  // web_header.html.twig

  {% set menus = web_service.getMenu(firstPart, secondPart) %}
  <ul>
      {% for item in menus.mainMenu %}
          <li><a id="{{ item.id }}" ... </a></li>
      {% endfor %}
  </ul>
  ```

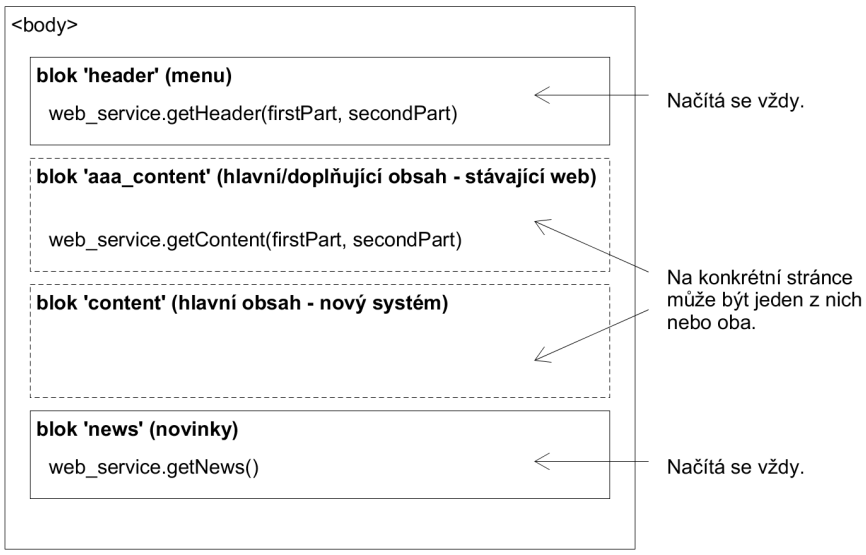
- *Stránky existujícího webu* – Na stránkách existujícího webu se (kromě předchozího) načítá hlavní obsah stránky.

  ```
  // web_content.html.twig

  {% block aaa_content %}
      {% set content = web_service.getContent(firstPart|default(null),
          secondPart|default(null)) %} // načtu obsah stránky
  ```

```
    {{ content|raw }} // zobrazení
{% endblock %}
```

- *Některé stránky nového systému* – Na některých stránkách nového systému se před hlavním obsahem stránky umístí část vytvořená zadavatelem. Jedná se např. o stránku s nabídkou produktů, kde chce mít zadavatel možnost upravit základní popis nabídky. Pro tuto funkčnost se používá blok `aaa_content` z předchozího příkladu, za nímž se nachází blok s hlavním obsahem. Blok `aaa_content` je na každé stránce, ale pokud nejsou v šabloně definovány proměnné `firstPart` a `secondPart`, nic nezobrazí. Díky tomu je možné umístit na libovolnou stránku obsah vytvořený zadavatelem, aniž by bylo potřeba měnit šablonu.



Obrázek 4.2: Integrace stávajícího webu

4.3.2 Administrační rozhraní

Administrační rozhraní bylo vytvořeno s pomocí balíčku *Sonata Admin*. Ten funguje na principu vytváření `Admin` tříd pro každou entitní třídu, jejíž data chceme v administraci spravovat. Tato `Admin` třída definuje 4 základní části rozhraní (každá z nich se definuje metodou, např. `configureListFields()`):

- *seznam* – zobrazení tabulky s přehledem instancí entity. V `Admin` třídě se definuje, které aktributy a jak mají být zobrazeny.

- *detail instance* – zobrazení detailu konkrétní instance.
- *filtry* – vlastnosti entit, podle kterých je možné instance filtrovat (např. uživatele podle země apod.).
- *formulář* – formulář pro editaci a vytváření instancí.

Konkrétní `Admin` třída vypadá např. takto (`LicenceAdmin`):

```
// LicenceAdmin.php

class LicenceAdmin extends BaseAdmin
{
    // konfigurace seznamu instancí
    protected function configureListFields(ListMapper $listMapper)
    {
        $listMapper
            ->addIdentifier('id')
            ->add('superLicence.id') // je možné zobrazit i atributy atributů
            ...
        ;
    }

    // konfigurace detailu instance
    protected function configureShowFields(ShowMapper $showMapper)
    {
        $showMapper
            ->add('id')
            ...
        ;
    }

    // konfigurace filtrů
    protected function configureDatagridFilters(DatagridMapper $datagridMapper)
    {
        $datagridMapper
            ->add('currentHWID.hwid', null, array(
                'label' => 'HWID'
            ))
            ...
        ;
    }

    // konfigurace formuláře
```

```
    protected function configureFormFields(FormMapper $formMapper)
    {
        $formMapper
            ->add('superLicence', 'entity', array(
                'class' => 'Simon\FurryBallBundle\Entity\SuperLicence',
                'property' => 'id',
            ))
            ->add('fbVersion', 'entity', array(
                'class' => 'Simon\FurryBallBundle\Entity\FBVersion',
                'property' => 'version',
            ))
           ...
        ;
    }
}
```

Všechny `Admin` třídy jsou zároveň definovány jako služby (viz 3.2.5), takže je možné do nich injektovat a používat ostatní služby a další parametry. Všechny `Admin` služby jsou definovány v konfiguračním souboru souboru `admin.yml` v balíčku `FurryBallBundle`.

Pro routování (viz 3.2.3) používají `Admin` třídy standardně controller `Sonata\AdminBundle\Controller\CRUDController`, ten je ale možné nahradit vlastním (poděděným) v konfiguraci služby, pokud funkčnost daná standardním controllerem nestačí.[23] Já takto například používám pro `EmailAdmin` vlastní `AdminEmailController`, obsahující metody (akce) pro posílání hromadných e-mailů, nebo také `AdminPurchaseController` s metodami pro schvalování objednávek.

4.3.3 Historie licencí

Jedním z požadavků na systém byla možnost zobrazit historii uživatelských změn licence, resp. SuperLicence – aktivace, upgrade, prodloužení apod. Pro to existuje více možných řešení.

Nejdřív jsem uvažoval o vytvoření kopie tabulky SuperLicence, kam by se při každé změně uložila „původní“ verze společně s časovým razítkem a typem změny. Toto řešení má ale několik nevýhod [25]:

- ukládám zbytečně i data, která se měnit nebudou, resp. jejich změny nepotřebuji zaznamenávat (např. uživatele nebo poznámku)

- naopak neukládám vazby, které jsou pro pro přehled historie důležité, např. na základě jaké objednávky došlo k prodloužení SuperLicence nebo jaká konkrétní licence vznikla upgradem na novou verzi.

Proto jsem se nakonec rozhodl vytvořit novou entitu Journal, jejíž instance budou sloužit pro záznam historie licencí. Tato entita obsahuje časové razítko, typ změny, vazbu na měněnou SuperLicenci další údaje (popis všech atributů je v tabulce 4.1). Slouží k záznamu 7 typů změn:

- *Nová SuperLicence* – záznam se vytvoří při vzniku SuperLicence.
- *Prodloužení* – záznam se vytvoří při prodloužení dočasné SuperLicence. Obsahuje údaje o tom od kdy do kdy prodloužení platí.
- *Upgrade* – záznam vznikne při upgradu trvalé SuperLicence na novější verzi. Obsahuje vazbu na novou licenci vzniklou upgradem.
- *Aktivace* – záznam vznikne při aktivaci licence. Obsahuje vazbu na aktivovanou licenci.
- *Změna HWID* – záznam vznikne při změně HWID.
- *Vydání nové verze* – záznam vznikne při automatickém upgradu trvalé SuperLicence po vydání nové verze FurryBall.
- *Prodloužení přednostní podpory* – záznam vznikne při prodloužení podpory trvalé SuperLicence.

V administračním rozhraní je pak v detailu každé SuperLicence přístupná její historie (seznam všech navázaných instancí Journalu seřazený podle času).

4.4 Bezpečnost

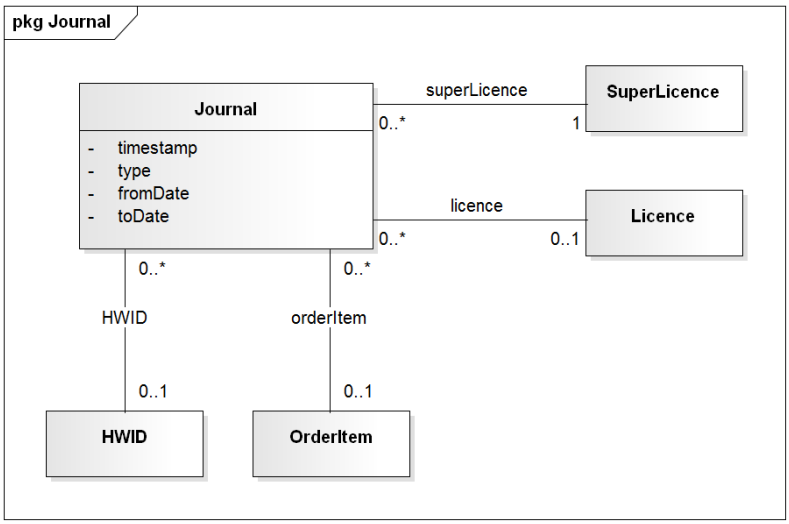
4.4.1 Základní rozdělení přístupu

Každá ze základních rolí v systému má jiná práva, a podle toho je rozdělen přístup k jednotlivým částem aplikace. Jde v základu o 3 sekce:

- *Administrátor* – pouze on má přístup k adresám začínajícím `/admin/*`. Kromě toho má přístup všude, kde běžný přihlášený uživatel.

Tabulka 4.1: Entita Journal

Atribut	Typ	Popis
id	int	identifikátor
timestamp	DateTime	časové razítko změny
type	int	typ změny (možné hodnoty jsou definovány jako třídní konstanty)
superLicence	SuperLicence	změněná SuperLicence
licence	Licence	změněná Licence (nepovinný údaj, použije se u *upgradu*, *aktivace*, *změně HWID* a *vydání nové verze*)
orderItem	OrderItem	položka objednávky, na základě které došlo ke změně (nepovinný údaj, použije se u *nové SuperLicence*, *prodloužení*, *upgradu* a *prodloužení podpory*)
hwid	HWID	HWID, které se účastní změny (nepovinný údaj, použije se u *aktivace* a *změny HWID*)
fromDate	DateTime	datum, od kterého je změna platná (nepovinný údaj, použije se u *prodloužení SuperLicence* a *prodloužení podpory*)
toDate	DateTime	datum, do kterého je změna platná (nepovinný údaj, použije se u *prodloužení SuperLicence* a *prodloužení podpory*)



Obrázek 4.3: Journal

- *Přihlášený uživatel* – má přístup k adresám začínajícím `/account/*`. Kromě toho má přístup všude, kde nepřihlášený uživatel.

- *Nepřihlášený uživatel* – má přístup k adresám, které nezačínají `/account/*` nebo `/admin/*`.

Nastavení přístupu se provede v souboru `/app/config/security.yml`:

```
// security.yml

security:
    access_control:
         - { path: ^/admin/, role: ROLE_ADMIN }
         - { path: ^/account/, role: ROLE_USER }
```

4.4.2 Ochrana proti základním typům útoků

4.4.2.1 SQL injection

Tento typ útoku využívá špatného ošetření uživatelských vstupů umožňující útočníkovi číst/upravovat data v databázi[26].

Komunikace s databází probíhá v aplikaci pomocí *Doctrine*. To standardně používá u všech SQL (resp. DQL) dotazů u předpřipravených funkcí parametry[27], což znemožňuje tento typ útoku. U vytváření vlastních DQL dotazů je třeba uživatelské vstupy vkládat pomocí funkce `QueryBuilder::setParameter()`, aby *Doctrine* poznalo, která část řetězce je uživatelským vstupem. V implementované aplikaci toto dodržuji.

4.4.2.2 Cross Site Scripting (XSS)

XSS umožňuje útočníkovi vložit do stránky vlastní kód (skript), který se poté spustí u každého uživatele, který stránku zobrazí[26].

V aplikaci používám šablonovací systém *Twig*, jehož jednou vlastností je i automatické ošetření výstupu, takže případný skript je interpretován jako prostý text (pro vložení textu bez ošetření je třeba použít filtr `|raw`).

4.4.2.3 Cross Site Request Forgery (CSRF)

Při CSRF útoku je z uživatelova prohlížeče odeslán (bez jeho vědomí) požadavek, tvářící se např. jako uživatelem vyplněný formulář z webu, na kterém je uživatel přihlášen (třeba internetové bankovnictví)[26].

Všechny formuláře v Symfony (pokud jsou vytvořeny pomocí komponenty `Forms`, viz. 3.3.1.5) jsou před tímto typem útoku automaticky chráněny pomocí skrytého formulářového pole `_csrf_token`.

Kapitola 5

Testování

5.1 Uživatelské testování

Pomocí uživatelského testování se zkoumá, jak s aplikací pracují skuteční uživatelé – zda je pro ně srozumitelná, funkce snadno dostupné a grafický návrh přehledný[32].

Pro testování jsem sestavil vstupní a výstupní dotazník a sadu scénářů, které bude každý uživatel procházet.

5.1.1 Vstupní dotazník

Vstupní dotazník vyplní každý každý testovaný uživatel před vykonáním testů.

1. Kolik je vám let?
2. Pohlaví
   a) muž
   b) žena
3. Vaše nejvyšší ukončené vzdělání:
   a) základní
   b) vyučen(a) / vyučen(a) s maturitou
   c) středoškolské
   d) vysokoškolské
4. Počítač používáte:
   a) denně
   b) několikrát týdně
   c) několikrát měsíčně

d) méně často

5. Vztah k modelovacímu a renderovacímu software (Maya, 3dsMax, Blender apod.):
   a) nepoužívám
   b) požívám ve volném čase
   c) používám profesionálně
6. Znalost renderovacího software FurryBall:
   a) neznám
   b) znám ale nepoužívám
   c) znám a používám
7. Zkušenost s webovými stránkami FurryBall:
   a) neznám
   b) znám ale nepoužívám často
   c) znám a používám často

5.1.2 Testovací scénáře

Testovací scénáře pokrývají základní činnosti, které uživatel potřebuje v aplikaci vykonávat. Testovány jsou pouze činnosti běžného uživatele, protože administrační rozhraní bylo vytvářeno pro konkrétní osoby a v přímé spolupráci s nimi.

- **Registrace**:

  Na stránkách `furryball.aaa-studio.eu` se registrujte s uživatelským jménem „test_user“, e-mailem „testuser@test.cz“ a heslem „test“.

  (*stav před*: V prohlížeči otevřená adresa `furryball.aaa-studio.eu`, *úspěch*: uživatel je registrován, *max. čas*: 10 minut)

- **Přihlášení a změna kontaktních informací**:

  1. Na stránkách `furryball.aaa-studio.eu` se přihlaste s uživatelským jménem „test_user“ a heslem „test“.
  2. Změňte své heslo na „nove_heslo“.
  3. Změňte svůj email na „novy_email@test.cz“ a jméno na „Jan Novák“.

  (*stav před*: V prohlížeči otevřená adresa `furryball.aaa-studio.eu`, *úspěch*: uživatel úspěšně změní a uloží nové kontaktní informace, *max. čas*: 10 minut)

- **Nákup licencí**:

  1. Na stránkách `furryball.aaa-studio.eu` najděte stránku s produkty.
  2. Vložte do košíku 2 licence typu „FurryBall 4 permanent“ a jednu typu „FurryBall 4 rental 6 months“.
  3. K jedné z permanentních licencí přidejte přednostní podporu („maintenance“).
  4. Vyplňte kontaktní informace.
  5. Za nákup zaplaťte platební kartou – číslo karty: 1111 1111 1111 1111, expirace: 01/2020, CCV: 111.

  (*stav před*: V prohlížeči otevřená adresa `furryball.aaa-studio.eu`, *úspěch*: uživatel úspěšně vytvoří objednávku, *max. čas*: 20 minut)

- **Správa licencí**:

  1. Na stránkách `furryball.aaa-studio.eu` se přihlaste s uživatelským jménem „test_user“ a heslem „test“.
  2. Najděte a otevřete stránku se svými licencemi.
  3. Mezi zobrazenými licencemi najděte jednu neaktivní a aktivujte ji (jako HWID zadejte libovolný řetězec).
  4. Mezi zobrazenými licencemi najděte jednu dočasnou a prodlužte ji (vložte její prodloužení do nákupního košíku).

  (*stav před*: V prohlížeči otevřená adresa `furryball.aaa-studio.eu`, *úspěch*: uživatel úspěšně aktivuje jednu licenci a objedná prodloužení druhé, *max. čas*: 10 minut)

5.1.3 Výstupní dotazník

Výstupní dotazník vyplní každý testovaný uživatel po vykonání testů. Spolu se záznamem aktivity uživatelů tvoří výstup testování.

1. Stránky jsou přehledné:
   a) zcela souhlasím
   b) spíše souhlasím
   c) spíše nesouhlasím
   d) zcela nesouhlasím
2. Hledané informace jsou snadno k nalezení:
   a) zcela souhlasím
   b) spíše souhlasím

c) spíše nesouhlasím
d) zcela nesouhlasím

3. Proces nákupu licence je srozumitelný:

a) zcela souhlasím
b) spíše souhlasím
c) spíše nesouhlasím
d) zcela nesouhlasím

4. Dal bych přednost původnímu procesu nákupu (pokud s ním máte zkušenosti):

a) zcela souhlasím
b) spíše souhlasím
c) spíše nesouhlasím
d) zcela nesouhlasím
e) nemám zkušenost

5. Stránka se správou mých licencí je užitečná:

a) zcela souhlasím
b) spíše souhlasím
c) spíše nesouhlasím
d) zcela nesouhlasím

Uvedené scénáře a vstupní/výstupní dotazník jsou připraveny jako podklady pro uživatelské testování části aplikace pro běžného uživatele (samotné testování není součástí této práce).

5.2 Funkční testování

Funkční testování zkoumá, zda se aplikace v dané situaci (při vykonávání konkrétního úkolu) chová správně. Tento typ testování prochází aplikaci z pohledu uživatele (simuluje uživatele). Mezi oblasti, které funkční testování zkoumá, patří např. ověřování funkčnosti formulářových prvků, reakce na nesprávný vstup uživatele nebo správné fungování odkazů. [28]

Aplikace v současnosti obsahuje tzv. *smoke testing* – testy zaměřené na ověření nejkritičtějších funkcí aplikace (např. jestli se aplikace vůbec spustí)[29]. Aplikace je rozdělena na 3 základní oblasti podle oprávnění uživatele:

- adresy dostupné všem uživatelům
- adresy dostupné přihlášeným uživatelům
- adresy dostupné administrátorům

U každé oblasti jsou otestována funkčnost (přístupnost) nejdůležitějších stránek. Dále aplikace obsahuje testy celého procesu nákupu (výběr a přidání produktů do košíku, vyplnění kontaktních údajů, potvrzení objednávky).

Testy se nacházejí ve složce `/src/Simon/FurryBallBundle/Tests`.

5.3 HTML validace

Validace stránky zkoumá, zda je její HTML kód napsaný v souladu s webovými standardy. Použité standardy jsou na stránce identifikovány pomocí deklarace `doctype` na jejím začátku. Navržená aplikace využívá prvky HTML5 (např. `data` atributy), proto je zvolen odpovídající `<!DOCTYPE HTML>`. K validaci byl použit validátor společnosti World Wide Web Consortium (W3C)[29]. W3C je na internetu autoritou, která se podílí na tvorbě webových standardů.

Při validaci byly odhaleny některé drobné problémy (chybějící atribut `type` u skriptu, duplikované `id`). Tyto problémy byly opraveny a v současnosti jsou stránky nového systému validní. Výjimkou jsou stránky administračního rozhraní, jejichž podoba je do značné míry dána použitou knihovnou (*SonataAdminBundle*, viz 3.1.1.5) a jejichž HTML kód obsahuje některé nestandardní atributy používané knihovnou.

5.4 Testování kompatibility

Při testování kompatibility je webová aplikace testována v různých internetových prohlížečích, aby se zjistilo, jestli se ve všech chová a zobrazuje stejně, příp. jaké jsou odchylky a zda je třeba aplikaci pro konkrétní prohlížeč upravit.

Jelikož je renderer FurryBall určený pro operační systém Windows 7 a novější, soustředil jsem se na nejpoužívanější prohlížeče na této platformě. Žebříčků nejpoužívanějších prohlížečů je mnoho[30][31], ale prvních 5 prohlížečů je vždy stejných (na desktopu), i když vzájemné pořadí se může lišit:

- Google Chrome
- Internet Explorer
- Firefox
- Safari
- Opera

Prohlížeč Safari je doménou Mac OS, testoval jsem tedy aplikaci na prohlížečích Google Chrome (verze 42.0.2311.90), Internet Explorer (verze 11.0.9600), Firefox (verze 37.0.2) a Opera (verze 28.0). V těchto prohlížečích se aplikace

[29] `https://validator.w3.org/`

chovala totožně. Objevily se pouze drobné vizuální odchylky vyplývající z rozdílných základních stylů jednotlivých prohlížečů u formulářových prvků. Tyto odchylky jsou ale skutečně minimální a nijak nenarušují rozvržení stránek ani použití aplikace.

Závěr

Cílem této práce bylo navržení a implementace webové aplikace pro prodej a správu licencí renderovacího softwaru FurryBall a pro podporu jeho uživatelů. Nejdříve bylo třeba analyzovat potřeby zadavatele a zformulovat funkční a nefunkční požadavky na navrhovanou aplikaci. Na jejich základě jsem poté navrhl základní strukturu systému a vybral vhodné technologie.

Jako základ pro aplikaci byl zvolen framework Symfony. To se v průběhu práce ukázalo být dobrým rozhodnutím – framework umožňoval soustředit se na vlastní business logiku aplikace a zároveň nekladl odpor, bylo-li třeba některé jeho části používat nestandardně.

Podařilo se implementovat všechnu navrženou funkčnost s výjimkou několika funkčních požadavků na komunikaci s desktopovým klientem (kapitola 2.4.1.3), resp. tyto požadavky jsou základně implementovány, ale nebyly nijak testovány. Všechny ostatní požadavky jsou plně funkční.

Pokud bych měl vybrat rozhodnutí v návrhu aplikace, které se ukázalo ne zcela optimálním, tak je to použití balíčku *FOSUserBundle*. Řešením problémů při jeho použití jsem strávil tolik času, že bych v něm pravděpodobně stihl implementovat vlastní balíček pro správu uživatelů. Nicméně toto mohu říct pouze zpětně, protože právě pomocí zkoumání zdrojového kódu tohoto balíčku jsem získal znalosti potřebné pro tvorbu vlastního řešení.

Místo pro zlepšení je určitě v optimalizaci aplikace, hlavně při komunikaci s databází – použití objektově-relačního mapování urychlilo vývoj a zpřehlednilo kód, na druhou stranu nebylo z časových důvodů při vývoji využito (vědomě) některých jeho technik, které optimalizují přístupy k databázi. Tuto optimalizaci je ale možno provést dodatečně a kromě datové vrstvy nijak nezasahuje do kódu aplikace.

Aplikaci se určitě bude v budoucnu rozšiřovat. Jednou z oblastí pro možné rozšíření je podrobnější analýza dat o uživatelích a licencích (systém v současnosti obsahuje pouze základní statistiky). Potřebná data a vazby jsou v databázi uloženy, jde tedy hlavně o vytvoření potřebné aplikační logiky.

Dalším rozšířením by mohlo být integrování existujícího WISIWIG editoru statických stránek webu (zmiňoval jsem o něm v kapitole 3.3.4) do nového administračního rozhraní – princip je stejný jako už existující integrace starého webu do nové aplikace.

Literatura

[1] Rouse, M.: LAMP (Linux, Apache, MySQL, PHP). [online], 2008, [cit. 2015-04-13]. Dostupné z: http://searchenterpriselinux.techtarget.com/definition/LAMP

[2] Stoupa, V.: Přehled a vývoj PHP frameworků. [online], 2008, [cit. 2015-04-13]. Dostupné z: http://www.root.cz/clanky/prehled-a-vyvoj-php-frameworku/

[3] SensioLabs: Projects using Symfony. [online], [cit. 2015-04-13]. Dostupné z: http://symfony.com/projects

[4] Tröster, F.: ORM frameworky pro PHP5: Obecný úvod. [online], 2010, [cit. 2015-04-13]. Dostupné z: http://www.zdrojak.cz/clanky/orm-frameworky-pro-php5-obecny-uvod/

[5] Ambler, S. W.: The Object-Relational Impedance Mismatch. [online], 2013, [cit. 2015-04-13]. Dostupné z: http://www.agiledata.org/essays/impedanceMismatch.html

[6] Getting Started With FOSUserBundle. [online], [cit. 2015-04-13]. Dostupné z: https://github.com/FriendsOfSymfony/FOSUserBundle/blob/master/Resources/doc/index.md

[7] Sonata Project. [online], [cit. 2015-04-13]. Dostupné z: https://sonata-project.org/

[8] Juras, M.: Výběr vhodného šablonovacího systému pro webintegrační projekt na platformě PHP. [online], 2014, [cit. 2015-04-13]. Dostupné z: http://www.web-integration.info/cs/blog/vyber-vhodneho-sablonovaciho-systemu-pro-webintegracni-projekt-na-platforme-php/

[9] SensioLabs: Twig for template designers. [online], [cit. 2015-04-13]. Dostupné z: http://twig.sensiolabs.org/doc/templates.html

[10] SensioLabs: Translations. [online], [cit. 2015-04-14]. Dostupné z: http://symfony.com/doc/current/book/translation.html

[11] SensioLabs: Projects using Symfony - phpBB. [online], [cit. 2015-04-14]. Dostupné z: http://symfony.com/projects/phpbb

[12] JavaScript. [online], 2014, [cit. 2015-04-14]. Dostupné z: https://developer.mozilla.org/cs/docs/Web/JavaScript

[13] jQuery Foundation: jQuery. [online], [cit. 2015-04-14]. Dostupné z: https://jquery.com/

[14] Michálek, M.: K čemu je dobrý Bootstrap a frontend frameworky? [online], 2013, [cit. 2015-04-14]. Dostupné z: http://www.zdrojak.cz/clanky/k-cemu-je-dobry-bootstrap-frontend-frameworky/

[15] Potencier, F.: What is Symfony2? [online], 2011, [cit. 2015-04-14]. Dostupné z: http://fabien.potencier.org/article/49/what-is-symfony2

[16] Fowler, M.: Front Controller. [online], [cit. 2015-04-15]. Dostupné z: http://martinfowler.com/eaaCatalog/frontController.html

[17] SensioLabs: Symfony internals: profiler. [online], [cit. 2015-04-15]. Dostupné z: http://symfony.com/doc/current/book/internals.html

[18] SensioLabs: Controllers. [online], [cit. 2015-04-15]. Dostupné z: http://symfony.com/doc/current/best_practices/controllers.html

[19] Grudl, D.: Co je Dependency Injection? [online], 2012, [cit. 2015-04-15]. Dostupné z: http://phpfashion.com/co-je-dependency-injection

[20] Fowler, M.: Inversion of Control Containers and the Dependency Injection pattern. [online], 2004, [cit. 2015-04-15]. Dostupné z: http://martinfowler.com/articles/injection.html

[21] SensioLabs: Service Container. [online], [cit. 2015-04-15]. Dostupné z: http://symfony.com/doc/current/book/service_container.html

[22] SensioLabs: Databases and Doctrine. [online], [cit. 2015-04-15]. Dostupné z: http://symfony.com/doc/current/book/doctrine.html

[23] Sonata Admin Architecture. [online], [cit. 2015-04-17]. Dostupné z: https://sonata-project.org/bundles/admin/2-3/doc/reference/architecture.html

[24] SensioLabs: How to Use Bundle Inheritance to Override Parts of a Bundle. [online], [cit. 2015-04-23]. Dostupné z: http://symfony.com/doc/current/cookbook/bundles/inheritance.html

[25] Downs, K.: History Tables. [online], 2008, [cit. 2015-04-23]. Dostupné z: http://database-programmer.blogspot.cz/2008/07/history-tables.html

[26] Ferschmann, P.: Bezpečnost na webu – přehled útoků na webové aplikace. [online], 2008, [cit. 2015-05-06]. Dostupné z: http://www.zdrojak.cz/clanky/prehled-utoku-na-webove-aplikace/

[27] Doctrine Security. [online], [cit. 2015-05-06]. Dostupné z: http://doctrine-orm.readthedocs.org/en/latest/reference/security.html

[28] Mazoch, B.: *Funkční testování webových aplikací*. Bakalářská práce, Masarykova univerzita, Brno, 2012, vedoucí práce Mgr. Tomáš Obšívač. Dostupné z: http://is.muni.cz/th/325233/fi_b/bmazoch-bp.pdf

[29] Smoke Testing. [online], 2011, [cit. 2015-04-27]. Dostupné z: http://softwaretestingfundamentals.com/smoke-testing/

[30] Top 5 desktop browsers Mar 2014 - Apr 2015. [online], 2015, [cit. 2015-04-26]. Dostupné z: http://gs.statcounter.com/#desktop-browser-ww-monthly-201403-201504

[31] Web browser Market Share, March 2015. [online], 2015, [cit. 2015-04-26]. Dostupné z: http://www.w3counter.com/globalstats.php?year=2015&month=3

[32] Fendrych, A.: Uživatelské testování návrhů webu. [online], 2009, [cit. 2015-05-06]. Dostupné z: http://www.lupa.cz/clanky/uzivatelske-testovani-navrhu-webu/

[33] How to Deploy a Symfony Application. [online], [cit. 2015-04-21]. Dostupné z: http://symfony.com/doc/current/cookbook/deployment/tools.html

Příloha A

Seznam použitých zkratek

CMS content management system.

CRUD create-read-update-delete.

CSRF Cross Site Request Forgery.

CSS cascading style sheet.

DOM Document Object Model.

DQL Doctrine Query Language.

HTML HyperText Markup Language.

JS Javascript.

MVC Model View Controller.

ORM objektově relační mapování.

SEO search engine optimization.

SQL Structured Query Language.

VPS virtual private server.

W3C World Wide Web Consortium.

XML Extensible Markup Language.

XSS Cross Site Scripting.

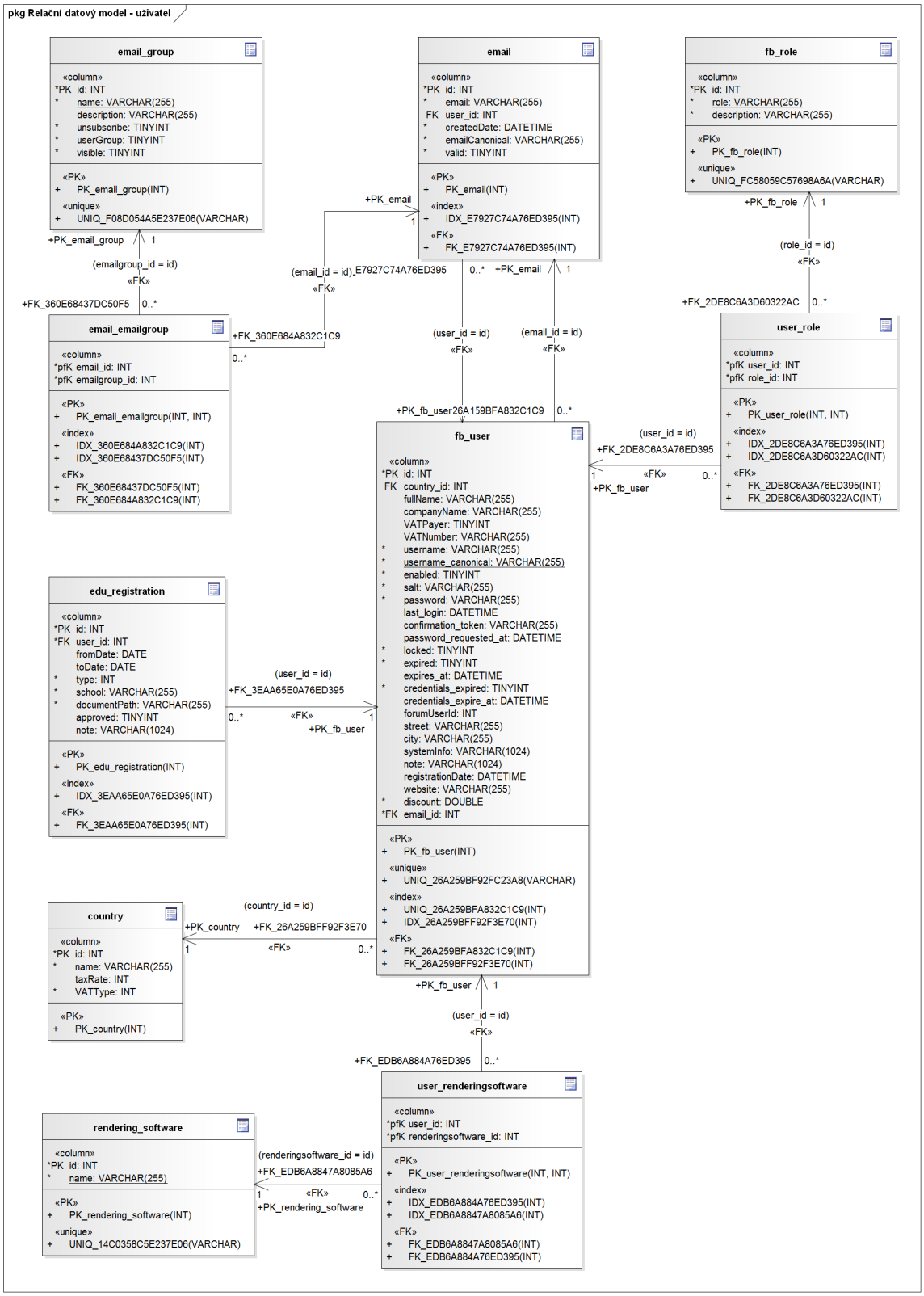
YAML YAML Ain't Markup Language.

Příloha B

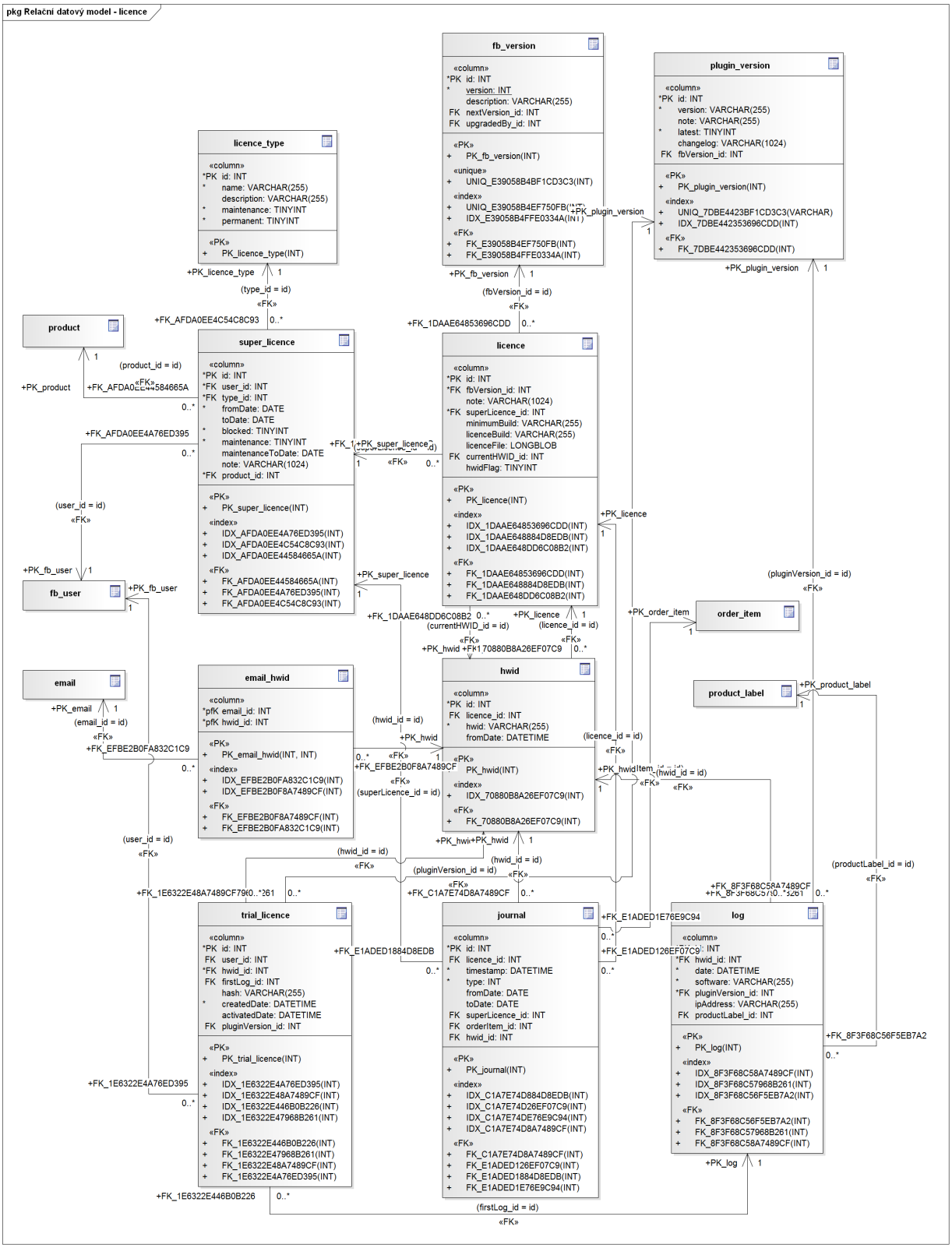
Struktura databáze

Tato příloha obsahuje diagramy databázové struktury systému. Pro přehlednost jsou tabulky rozděleny do tří základních skupin – *uživatel*, *licence* a *nákup* (tabulky z jiné skupiny jsou ve vazbách znázorněny pouze názvem).

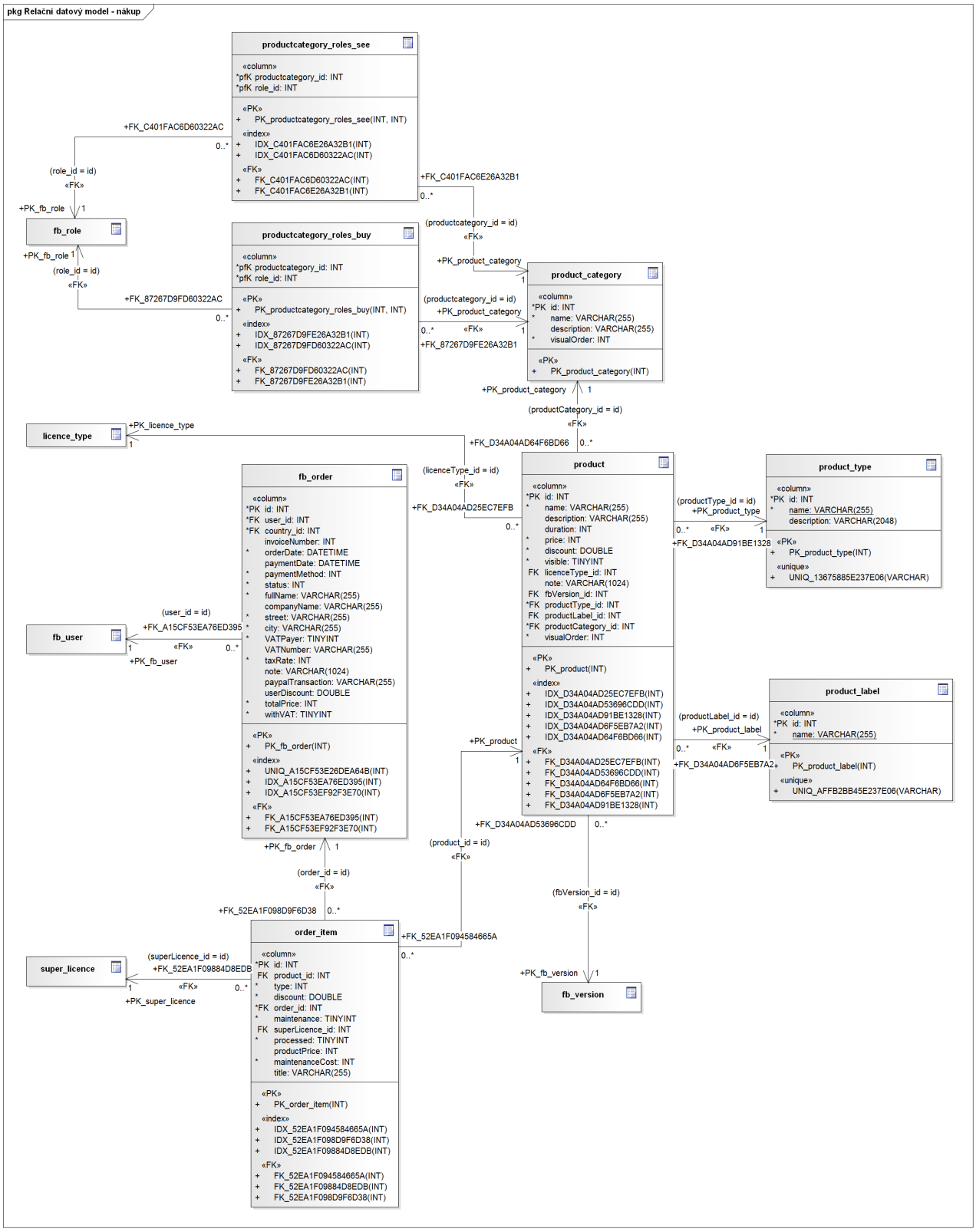
Diagramy neobsahují tabulky původního systému, které jsou zcela nezávislé a navrhovaný systém s nimi komunikuje pouze pomocí služeb z balíčku `AAAStudio`.



Obrázek B.1: Relační datový model – uživatel



Obrázek B.2: Relační datový model – licence



Obrázek B.3: Relační datový model – nákup

Příloha C

Instalační příručka

Tato příručka popisuje postup instalace (nasazení) navrženého systému na produkční server. Tento postup do značné míry vychází ze standardního způsobu nasazení aplikací vytvořených ve frameworku Symfony. [33]

Jelikož jedním z nefunkčních požadavků (požadavek N6.) bylo nasazení na konkrétní hosting, používající LAMP stack (Linux, Apache, MySQL, PHP) [1], je tato příručka určena primárně pro nasazení na Linux s databází MySQL, ale postup nasazení např. na Windows by byl v podstatě stejný.

C.1 Nasazení na produkční server

Při nasazení systému postupujeme podle následujících kroků:

1. *Nahrání zdrojových souborů projektu na server* – jako první je třeba nahrát na server zdrojový kód systému. To je možné dvěma způsoby:

   - Zkopírováním obsahu adresáře `/src` z CD, které je přílohou této práce, do zvoleného umístění na serveru.
   - Naklonováním Git[30] repozitáře projektu do zvoleného umístění na serveru (pokud máte k repozitáři přístupová práva) pomocí příkazu:

     ```
     $ git clone https://steklsim@bitbucket.org/steklsim/furryball.git
     ```

2. *Kontrola vlastností serveru* – Symfony umožňuje zkontrolovat, zda nastavení serveru splňuje požadavky pro fungování frameworku. Kontrola se provede příkazem:

[30] systém pro správu verzí

```
$ php app/check.php
```

3. *Import dat do databáze* – struktura a základní data databáze jsou v souboru `/resources/database_structure.sql` na CD, které je přílohou této práce. Pokud databáze nebyla už dříve importována, je třeba tato data importovat (např. pomocí *phpMyAdmin*).

4. *Stažení závislostí* – zdrojový kód je na CD i v repozitáři bez závislostí (knihoven třetích stran), ty jsou definovány v souboru `composer.json`. Pokud na serveru není instalován *Composer*[31], je třeba stáhnout ze stránky `https://getcomposer.org/download/` soubor `composer.phar` a umístit ho rootu projektu na serveru. Poté se závislosti stáhnou příkazem (spuštěným z rootu projektu):

   ```
   $ php composer.phar install
   ```

   (resp. `$ composer install`, pokud je *Composer* na server nainstalován)

   V průběhu tohoto příkazu se také systém zeptá na hodnoty parametrů, které potřebuje pro své fungování (údaje pro připojení k databázi, nastavení pro posílání e-mailů a další parametry specifické pro tuto aplikaci). Tyto parametry se poté uloží do souboru `/app/config/parameters.yml` (kde je můžete později změnit), jejich popis najdete v souboru `/app/config/parameters.yml.dist`.

5. *Přidání uživatele - admina* – v tuto chvíli nejsou v systému žádní uživatelé. Je potřeba přidat uživatele - administrátora, aby bylo možno používat administrátorské rozhraní. To se provede tímto příkazem (parametry `login`, `email` a `heslo` nahraďte svými hodnotami):

   ```
   $ php app/console fos:user:create login email heslo --super-admin
   ```

6. *Cache* – Symfony používá pro některé části aplikace cache (např. Twig šablony jsou zde zkompilovány do PHP šablon). Pro její vyčištění a opětovné vygenerování se použije příkaz:

   ```
   $ php app/console cache:clear --env=prod --no-debug
   ```

[31] nástroj pro správu závislostí

7. *Příprava statických zdrojů* – některé statické zdroje (hlavně CSS, JS soubory) knihoven třetích stran se nacházejí v jejich adresářích. Je třeba je zkopírovat do adresáře `/web` a zde provést jejich optimalizaci (např. zkombinováním více souborů do jednoho) a další úpravy před jejich použitím. Pro tuto činnost je použita knihovna *Assetic* (viz kapitola 3.1.1.8). Statické zdroje se připraví pomocí příkazů:

```
$ php app/console assets:install --symlink
$ php app/console assetic:dump --env=prod --no-debug
```

Po provedení těchto kroků by měl být systém funkční. Můžete se přihlásit pomocí údajů použitých v kroku 5.

C.2 Konfigurace

Konfigurace se provádí editací několika základních konfiguračních souborů. Tyto soubory se dají rozdělit do dvou kategorií:

- *konfigurace celé aplikace* – konfigurační soubory pro celou aplikaci. Nacházejí se ve složce `/app/config/` a jsou to:
  - `config.yml` – základní konfigurační soubor, do kterého jsou importovány ostatní konfigurační soubory. Jsou zde nastavení pro samotný framework, *Twig*, *Assetic*, *Doctrine*, *Swiftmailer*[32] a další části aplikace.
  - `parameters.yml` – zde jsou parametry pro připojení k databázi, e-mailovému účtu a také parametry specifické pro tuto konkrétní aplikaci, jako např. umístění a URL podpůrného fóra.
  - `routing.yml` – v tomto souboru je definováno mapování URL adres na konkrétní metody controllerů. Jelikož v této aplikaci je mapování definováno pomocí anotací přímo ve zdrojovém kódu controllerů, jsou zde pouze tyto zdrojové soubory importovány. Pokud nechcete měnit funkčnost aplikace, není třeba v tomto souboru nic měnit.
  - `security.yml` – zde jsou bezpečnostní nastavení aplikace – jak a kde jsou uloženi uživatelé, přístup do jednotlivých částí aplikace na základě rolí apod.
- *konfigurace balíčku FurryBallBundle* – konfigurace specifická pro tento balíček se nachází ve složce `/src/Simon/FurryBallBundle/Resources/config`. Jde o tyto soubory:

[32] knihovna pro posílání e-mailů

- `services.yml` – definice služeb (viz 3.2.5) balíčku, kromě `Admin` služeb.
- `admin.yml` – definice `Admin` služeb (viz 4.3.2).

Příloha D

Uživatelská příručka

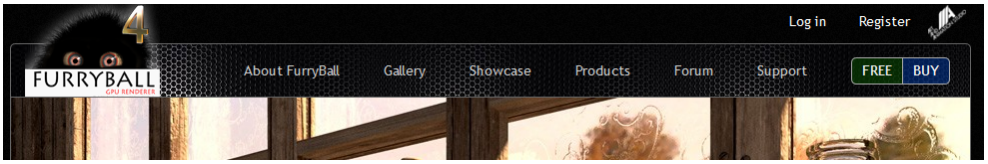
D.1 Běžný uživatel

Tato část uživatelské příručky slouží k seznámení se systémem z pohledu běžného uživatele. Obsahuje popis základních činností, se kterými se uživatel při používání systému setká. U každé činnosti je kromě popisu i obrázek odpovídající webové stránky.

D.1.1 Registrace a přihlášení

D.1.1.1 Registrace

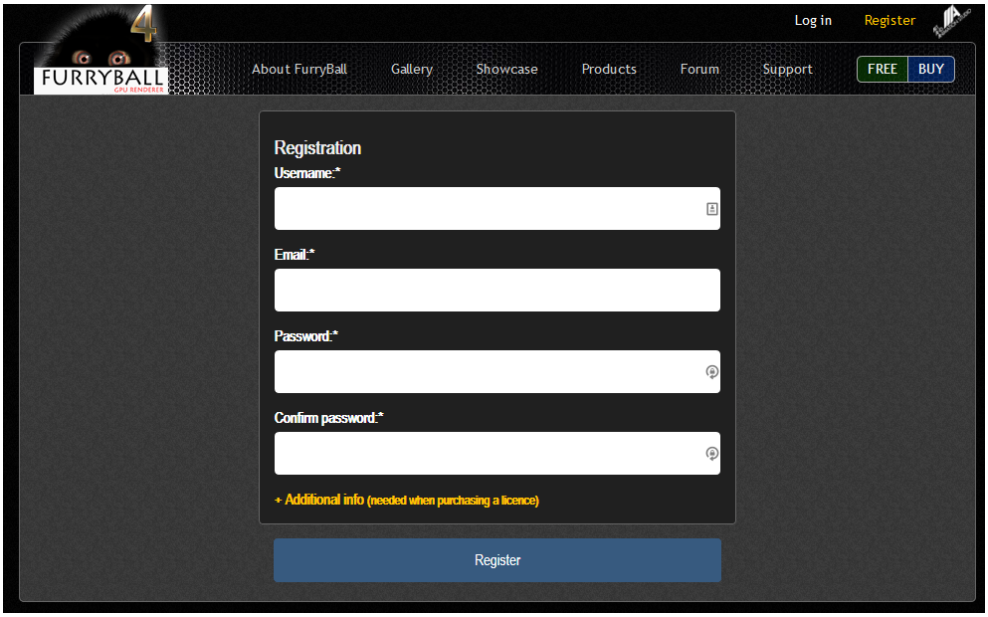
Pro registraci slouží odkaz „Register“ v menu na horním okraji stránky (dále jen „horní menu“):



Obrázek D.1: Odkaz na registraci/přihlášení

Při registraci je třeba vyplnit uživatelské jméno, e-mail a heslo. Po rozkliknutí je možné vyplnit i další údaje (celé jméno, adresa, . . .), které při registraci nejsou povinné, ale je třeba je vyplnit před dokončením nákupu.

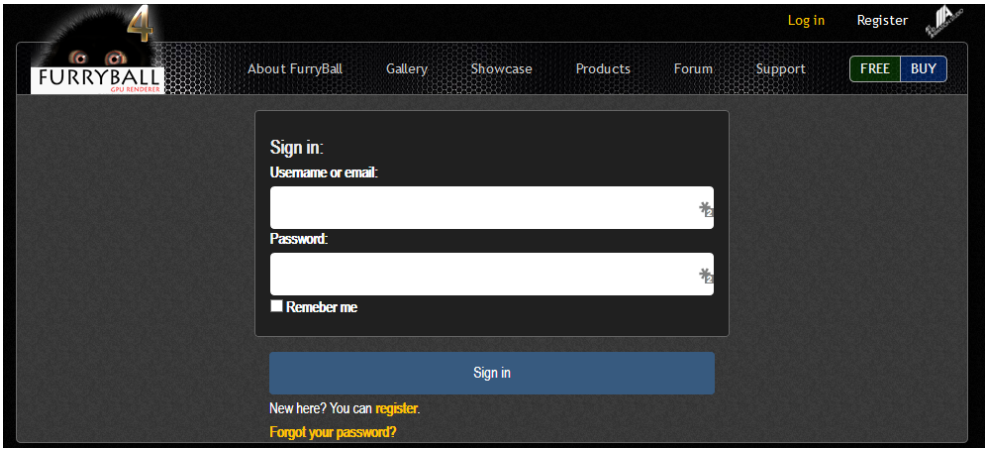
Po odeslání registračního formuláře se zobrazí stránka o úspěšné registraci (pokud je vše v pořádku) a je možné okamžitě používat funkce uživatelského účtu.



Obrázek D.2: Registrační formulář

D.1.1.2 Přihlášení

Přihlásit se lze po kliknutí na odkaz „Log in“ vyplněním uživatelského jména nebo e-mailu a hesla:



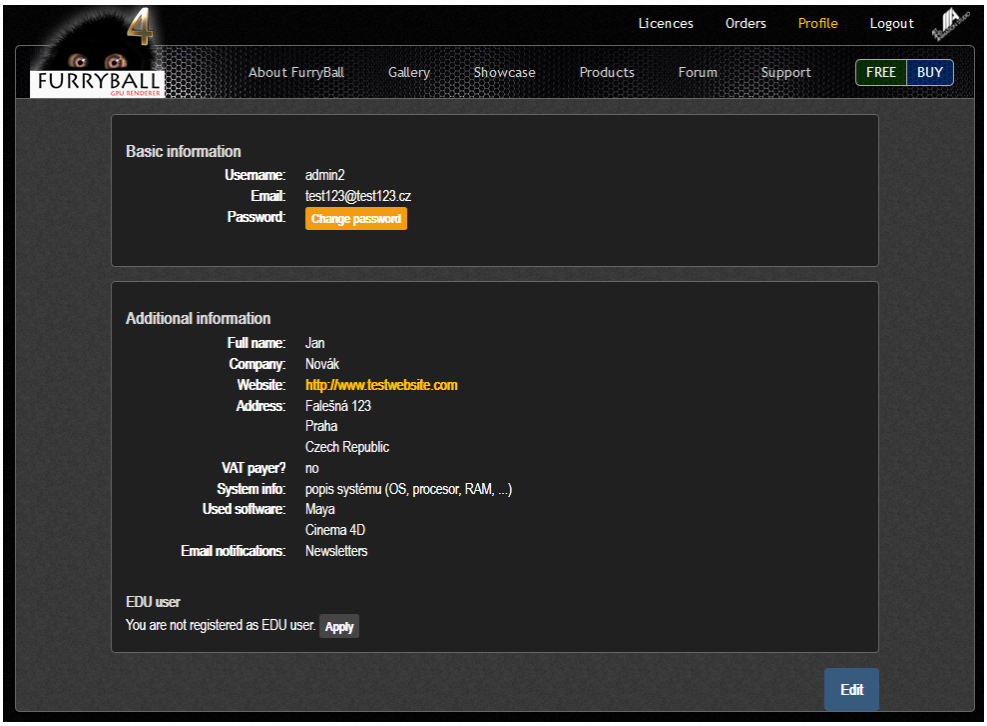
Obrázek D.3: Přihlášení

Pokud zapomenete heslo, vyberte odkaz „Forgot your password?“. Po zadání e-mailu, který jste použili při registraci bude na tento e-mail odeslán odkaz umožňující změnu hesla.

D.1.2 Profil

D.1.2.1 Zobrazení profilu

Pro zobrazení profilu použijte odkaz „Profile“ v horním menu. Profil obsahuje vaše kontaktní informace (e-mail, uživatelské jméno, adresu, ...) a také doplňující informace – s jakým softwarem FurryBall používáte, nastavení upozornění a zda jste EDU uživatelem.



Obrázek D.4: Profil

D.1.2.2 Editace profilu

Pro úpravu profilových informací použijte tlačítko „Edit“ na dolním okraji stránky. Zobrazí se obdobný formulář jako při registraci (pouze bez uživatelského jména). Změny potvrdíte tlačítkem „Save changes“.

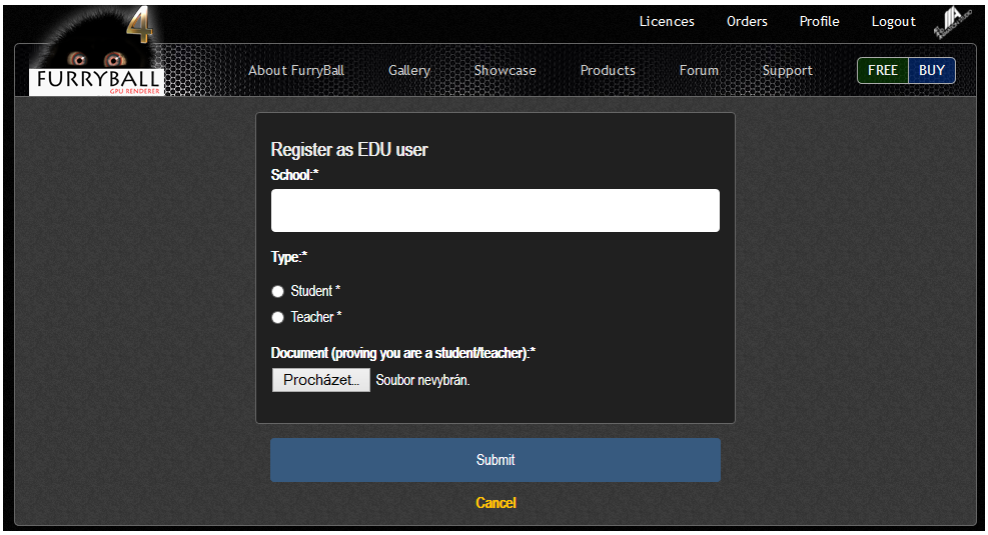
Pro změnu hesla použijte tlačítko „Change password“. Do zobrazeného formuláře vyplníte nejdříve staré heslo, poté 2x nové (pro potvrzení) a nakonec změnu uložíte pomocí tlačítka „Change password“.

D.1.2.3 EDU registrace

Pokud jste studentem nebo učitelem, můžete se registrovat jako EDU uživatel – budete pak mít možnost kupovat speciální EDU produkty. Ty jsou cenově

výhodnější, na druhou stranu obsahují některá omezení.

Pro vyplnění žádosti o EDU registraci použijte v profilu v sekci „EDU user“ odkaz „Apply“. Zobrazí se formulář, který je vidět na obrázku D.5 a kde je třeba vyplnit jméno vaší školy, zda jste student nebo učitel a přiložit fotku/scan dokumentu dokazujícího, že jste studentem/učitelem (např. školní karta, index, ...).



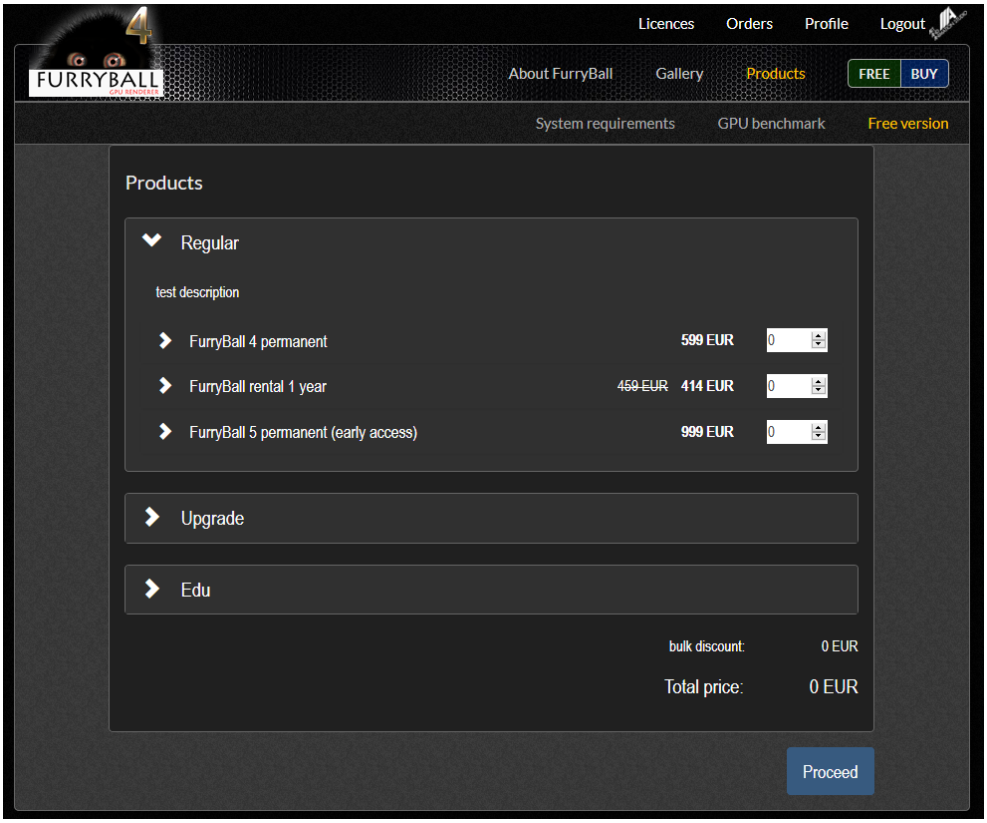
Obrázek D.5: EDU registrace

Po odeslání formuláře musí být registrace schválena administrátorem (cca 1-2 dny). Na schválení budete upozorněni e-mailem a v profilu v sekci „EDU user“.

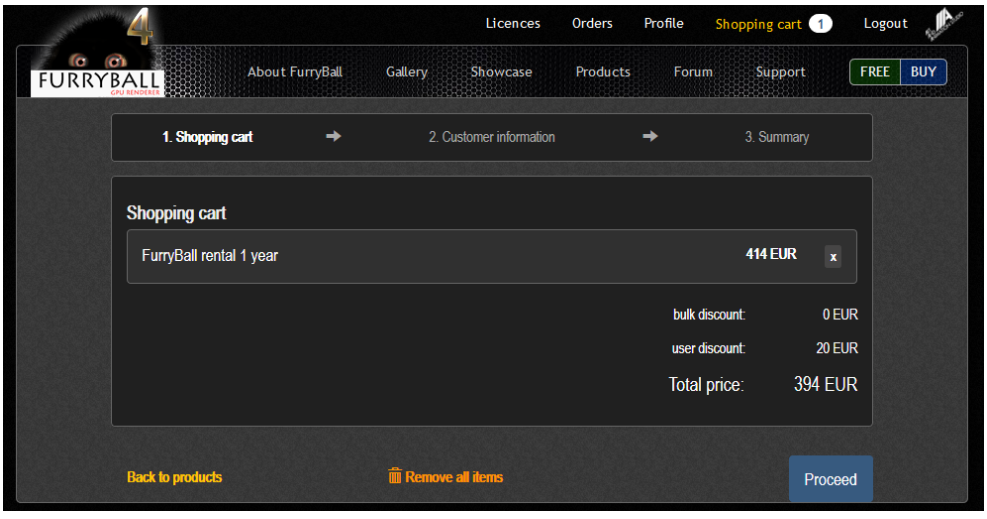
D.1.3 Nákup

Nákup licencí funguje obdobným způsobem jako v klasických e-shopech. Pro zobrazení produktů použijte odkaz „Products“ v hlavním menu. Zobrazí se obrazovka jako na obrázku D.6, kde zadejte typ a množství požadovaných licencí (detailní popisy produktů lze zobrazit kliknutím na jejich název, stejným způsobem lze „rozbalit“ sbalené kategorie produktů). Na dolním okraji stránky se zobrazuje celková cena a příp. množstevní sleva (při nákupu více licencí stejného typu).

Po vybrání produktů pokračujte stiskem tlačítka „Proceed“. Zobrazí se nákupní košík se všemi vybranými produkty (obrázek D.7). Zde je možné k některým licencím přidat přednostní podporu (za poplatek máte přednost při řešení problémů a po dobu platnosti podpory dostanete všechny nově vydané



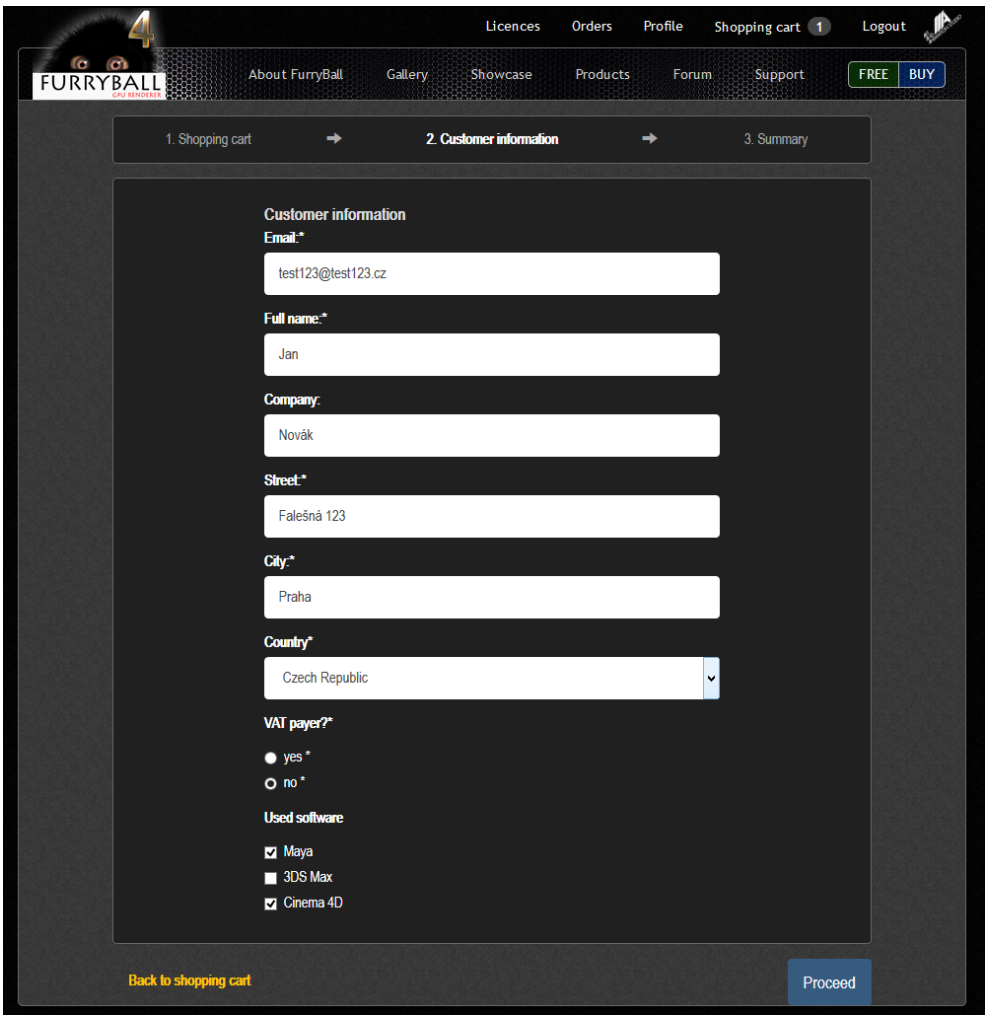
Obrázek D.6: Nabídka produktů



Obrázek D.7: Nákupní košík

verze zdarma). Je také možné jednotlivé položky odebrat, příp. vyprázdnit celý nákupní košík (odkaz „Remove all items“).

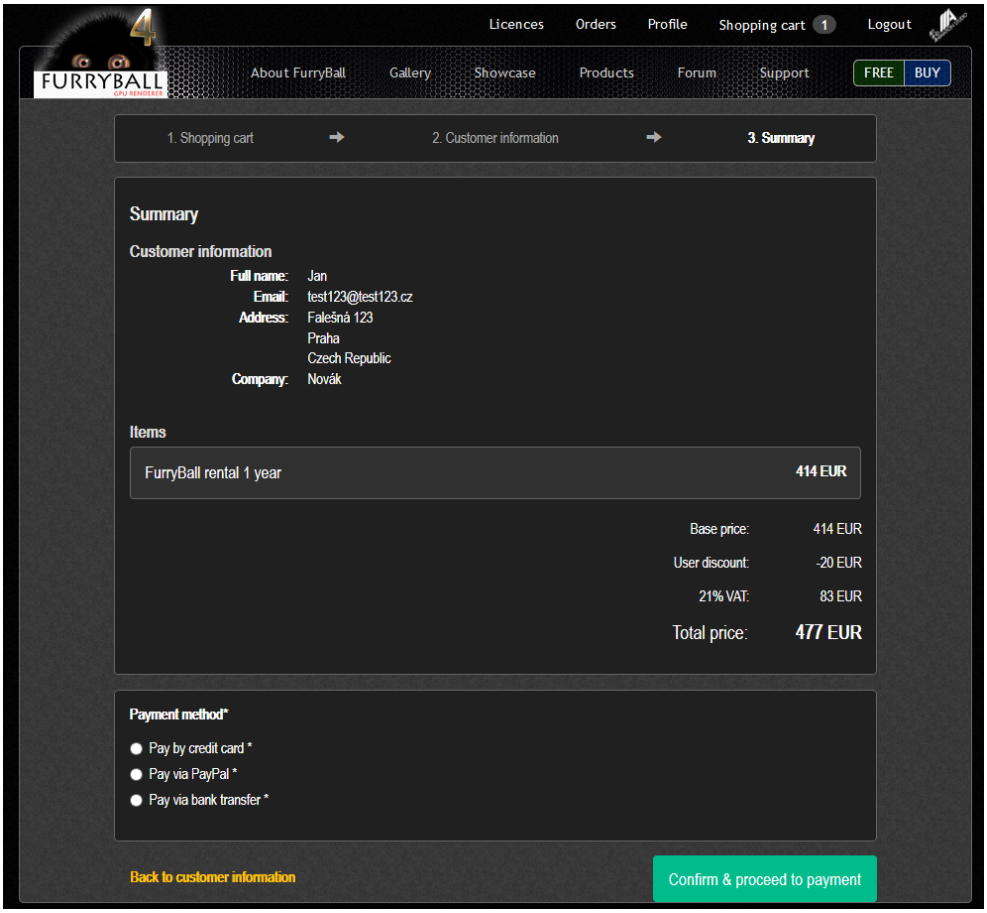
Pro pokračování stiskněte tlačítko „Proceed“. Zobrazí se formulář s kontaktními informacemi (pokud jste tyto informace už dříve vyplnili při registraci nebo v profilu, budou předvyplněné), který je vidět na obrázku D.8.



Obrázek D.8: Nákup – kontaktní informace

Po vyplnění formuláře ho odešlete tlačítkem „Proceed“. Zobrazí se souhrn objednávky s kontaktními informacemi, položkami objednávky a celkovou cenou. Pod souhrnem vyberte způsob platby. Po výběru platební metody potvrdíte celou objednávku tlačítkem „Confirm & proceed to payment“.

Pokud jste vybrali platbu kartou nebo přes PayPal, budete přesměrováni



Obrázek D.9: Nákup – souhrn objednávky

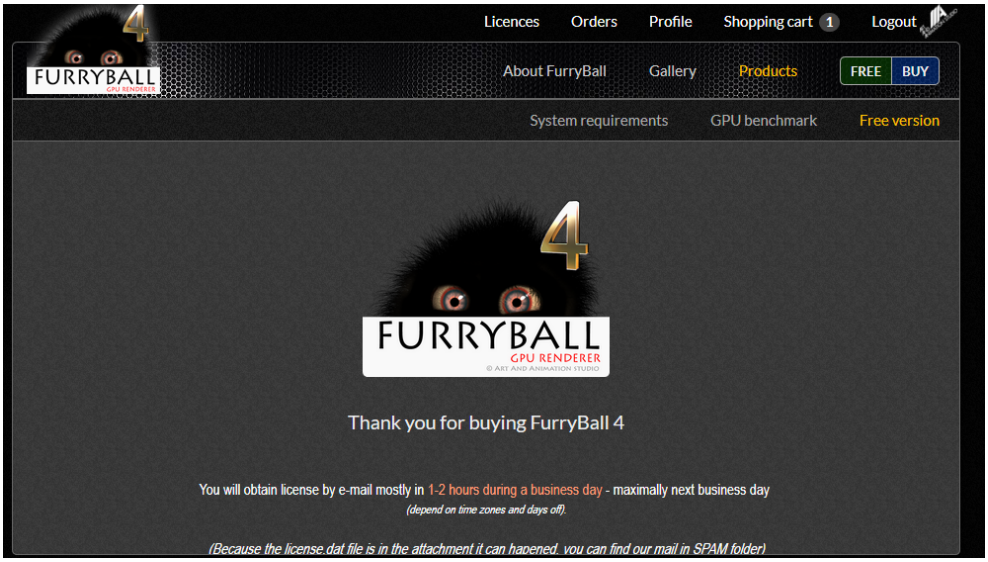
na danou platební bránu. Po úspěšné platbě budete informování e-mailem a obdobnou stránkou jako na obrázku D.10.

Pokud jste vybrali platbu převodem, zobrazí se stránka s informacemi kam peníze poslat.

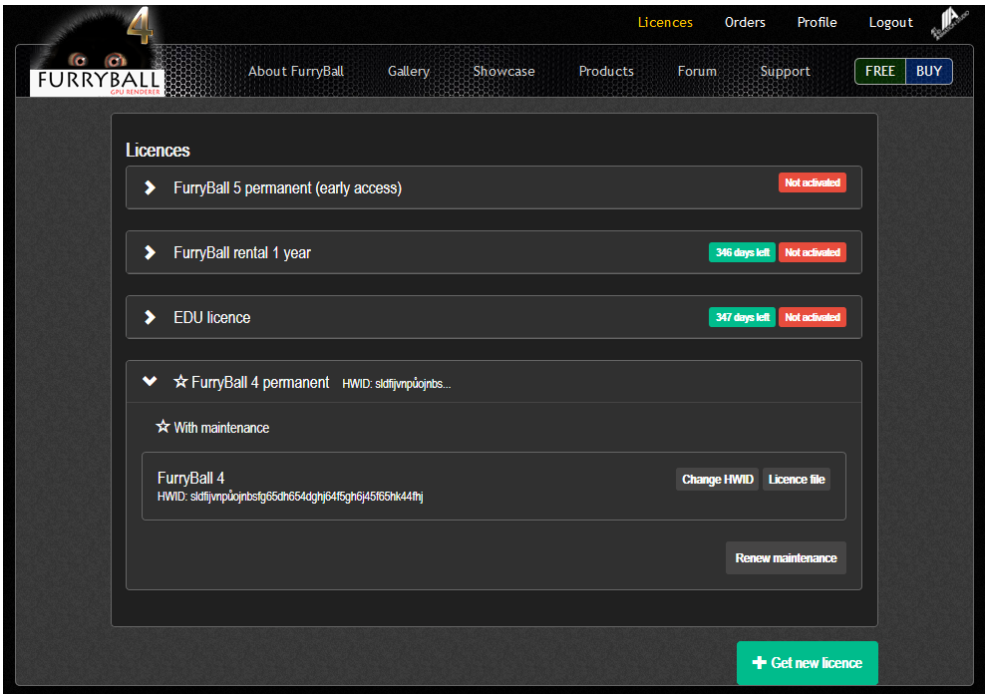
D.1.4 Správa licencí

D.1.4.1 Přehled licencí

Přehled licencí je pro přihlášeného uživatele dostupný přes odkaz „Licences“ v horním menu. Přehled obsahuje informace o všech vašich licencích – jejich typ, platnost apod. (viz obrázek D.11).



Obrázek D.10: Nákup – potvrzení úspěšné objednávky



Obrázek D.11: Přehled licencí

D.1.4.2 Úpravy licence

U každé licence je po rozkliknutí jedna nebo více možností jejich úpravy.

- *Aktivace* – každou novou licenci je třeba před použitím aktivovat (přiřadit jí platné HWID). To se provede tlačítkem „Activate“. V následujícím formuláři zadáte HWID vygenerované vaším desktopovým klientem FurryBall.
- *Změna WHID* – u každé licence je možné několikrát ročně změnit HWID. Provede se to tlačítkem „Change HWID“, zbytek postupu je stejný jako u aktivace. Pokud je tato tlačítko neaktivní, znamená to, že jste vyčerpali všechny možnosti změny HWID. Pokud nutně potřebujete HWID změnit, napiště na podporu FurryBall.
- *Upgrade* – u permanentních licencí je možnost provést upgrade na nejnovější verzi (dočasné licence jsou upgradovány automaticky). Tlačítkem „Upgrade licence“ vložíte upgrade do nákupního košíku a poté pokračujte podle kapitoly D.1.3.
- *Prodloužení podpory* – u permanentních licencí je možné prodloužit přednostní podporu o další rok. Tlačítkem „Renew maintenance“ vložíte toto prodloužení do košíku a dále pokračujte podle kapitoly D.1.3.
- *Prodloužení licence* – u dočasných licencí je možnost jejich prodloužení. To se provede tlačítkem „Renew licence“. Do košíku bude vloženo prodloužení licence o dobu její platnosti (např. půlroční bude prodloužena o další půlrok). Dále pokračujte podle kapitoly D.1.3.
- *Licenční soubor* – pro stažení licenčního souboru použijte tlačítko „Licence file“.

D.1.5 Přehled objednávek

Přehled objednávek zobrazíte pomocí odkazu „Orders“ v horním menu. Tato sekce obsahuje informace o všech vašich objednávkách – jejich stav, přehled položek a cenu.

D.2 Administrátor

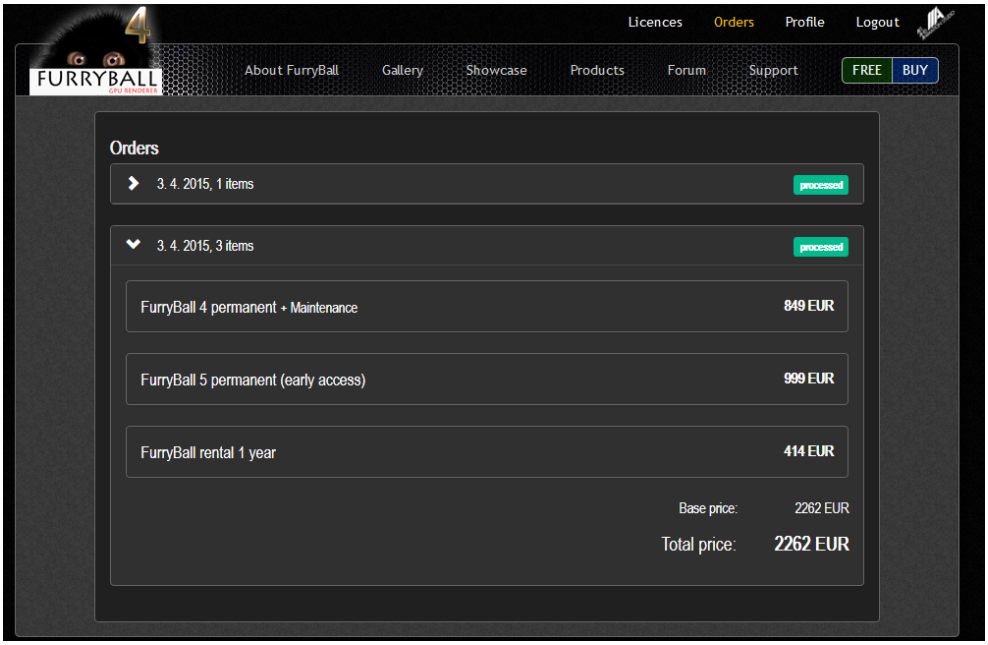
Tato část uživatelské příručky slouží k seznámení s administrátorským rozhraním. Obsahuje popis principů jeho fungování a dále popisy jednotlivých stránek (důležité funkce jsou rozepsány do jednotlivých kroků).

D.2.1 Přístup do administračního rozhraní

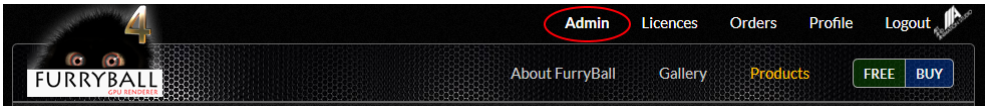
Do administračního rozhraní se dostanete po přihlášení kliknutím na odkaz „Admin“ v horním menu (odkaz se zobrazí jen pokud máte administrátorská práva).

D.2.2 Základní rozvržení

Každá obrazovka administračního rozhraní se skládá ze tří základních částí:



Obrázek D.12: Přehled objednávek

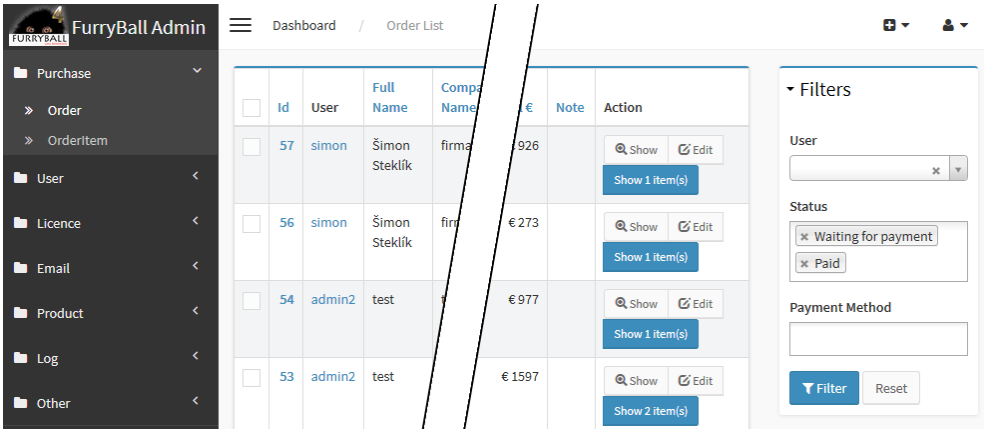


Obrázek D.13: Vstup do administrace

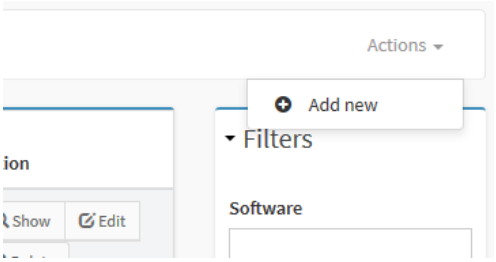
- *Navigační panel (vlevo)* – obsahuje odkazy na všechny obrazovky administrace.
- *Hlavní obsah (uprostřed)* – většinou tabulka s přehledem dat (např. přehled produktů, licencí apod.). Na konci každého řádku tabulky jsou akce, které je možné s daným řádkem (instancí) provést (zobrazit detail, editovat apod.). Na spodním okraji tabulky vlevo je nabídka hromadných akcí – nejdříve v tabulce zaškrtnete, kterých řádků se má akce týkat a poté jí dole vyberete a potvrdíte.
- *Filtry (vpravo)* – slouží k filtrování dat podle jejich vlastností (např. objednávky podle stavu).

Každá spravovaná entita (která má vlastní obrazovku administračního rozhraní) má 3 základní zobrazení:

- *Přehled* – seznam (tabulka) všech instancí.
- *Detail* – detailní zobrazení konkrétní instance. Zobrazí se pomocí odkazu „Show“ u daného řádku v Přehledu.



Obrázek D.14: Základní rozvržení administrace



Obrázek D.15: Vytvoření nové instance

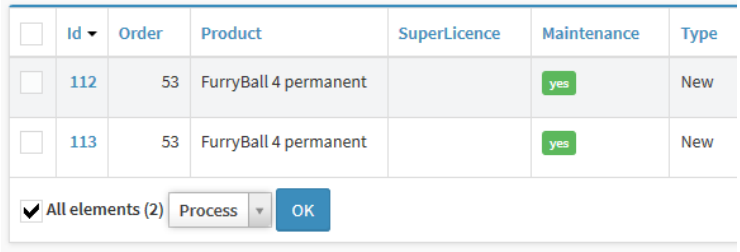
- *Formulář* – formulář pro editaci a vytváření nových instancí. Zobrazí se pomocí odkazu „Edit“ u daného řádku v Přehledu (pokud chcete editovat existující instanci), nebo kliknutím na „Actions->Add new“ v pravém horním rohu Přehledu.

D.2.3 Přehled obrazovek

D.2.3.1 Orders, Order Items (objednávky)

Stránka s přehledem a zpracováním objednávek. Zpracování objednávky se provede takto:

1. U dané objednávky kliknete na odkaz „Show items“.
2. V zobrazeném přehledu položek objednávky zaškrtnete vlevo dole „All elements“ (tím vyberete všechny zobrazené položky), vedle zvolíte akci "Process"a potvrdíte kliknutím na tlačítko „Ok“ (viz obrázek D.16).
3. Aplikace poté automaticky zpracuje položky objednávky (vytvoří/upraví licence) a informuje vás o výsledku.



Obrázek D.16: Zpracování objednávky

D.2.3.2 Users (uživatelé)

Základní přehled uživatelů. Je zde možnost zobrazit licence konkrétního uživatele (odkaz „Superlicences“), a dále je možné vybraným uživatelům poslat hromadný e-mail. To se provede pomocí hromadné akce „Send e-mail“:

1. Zaškrtnete zvolené uživatele.
2. Zvolíte a potvrdíte hromadnou akci „Send e-mail“.
3. Zobrazí se formulář pro odeslání e-mailu s předvyplněnými adresami (obrázek D.17). Zde buď vyberete jednu z uložených zpráv, nebo vytvoříte vlastní.
4. Po odeslání formuláře budete informováni o výsledku.

D.2.3.3 User statistics (statistiky)

Tato obrazovka zobrazuje některé detailní informace o uživatelích – kolik mají licencí jakého typu, neaktivních licencí apod.

D.2.3.4 EDU Registrations

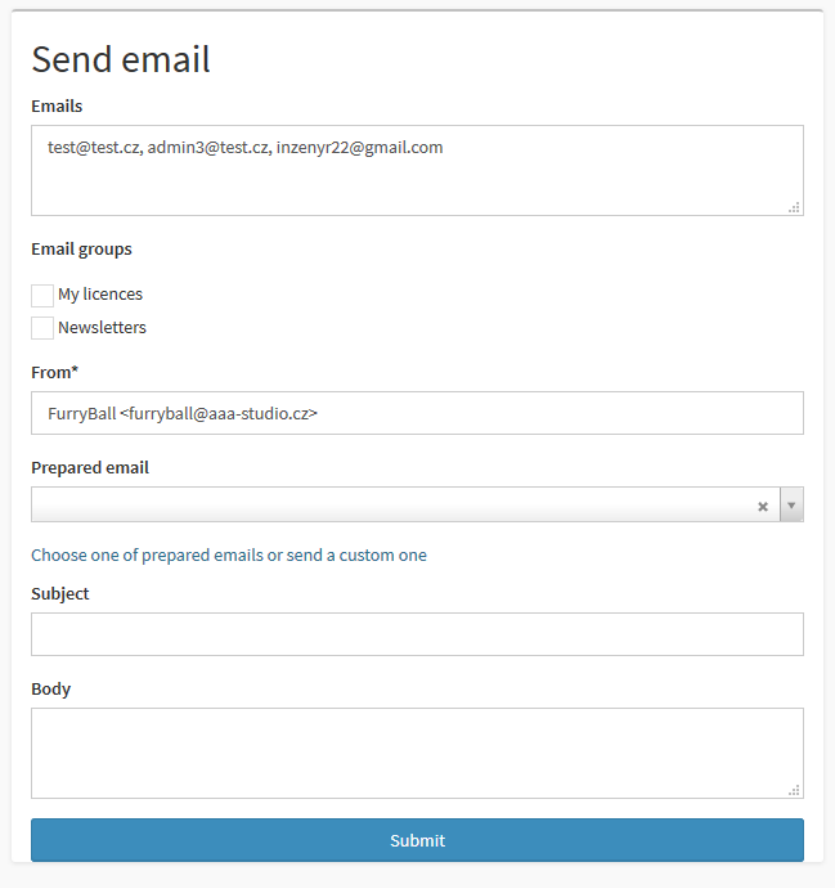
Zde je přehled všech EDU registrací s možností zkontrolovat a schválit (odkaz „Approve“). Je zde také možnost stáhnout si dokument přiložený k registraci (oskenovaný index, školní karta).

D.2.3.5 Superlicences, Licences

Přehled superlicencí s možností zobrazení přiřazených licencí (odkaz „Show licences“).

D.2.3.6 Trial Licences

Přehled zkušebních licencí.



Obrázek D.17: Odeslání hromadného e-mailu

D.2.3.7 Emails

Přehled a správa e-mailových adres. Je možné odeslat hromadný e-mail na více adres (postup je stejný jako u uživatelů, viz D.2.3.2). Dále je možné hromadně přidat e-mailové adresy do určité skupiny (hromadná akce „Add to group“, resp. „Remove from group“, ve vedlejším poli se vybere daná skupina).

D.2.3.8 Email Groups

Přehled a správa skupin e-mailových adres. Tlačítkem „Send email“ se odesílá hromadná zpráva všem adresám v dané skupině (zobrazí se formulář pro odeslání jako na obrázku **??**).

D.2.3.9 Products, Product Categories, Product Labels

Přehled a správa produktů a jejich kategorií (kategorie určuje, kdo může daný produkt koupit/vidět). *Product Labels* je číselník interních názvů produktů

jak jsou používány desktopovými klienty FurryBall při hlášení stavu.

D.2.3.10 Logs

Přehled hlášení desktopových klientů FurryBall.

D.2.3.11 Roles

Přehled a správa rolí uživatelů. Umožňuje přidávat nové role a upravovat stávající (kromě systémových, které jsou interně používány aplikací a nelze je proto měnit).

D.2.3.12 FB Versions

Přehled a správa major verzí FurryBallu (permanentní produkty/licence jsou vždy vázány na konkrétní verzi, např. produkt „FurryBall 4 permanent“ je vázán na verzi 4 a pro použití FurryBallu 5 je třeba provést upgrade).

D.2.3.13 Plugin Versions

Přehled a správa verzí renderovacího pluginu FurryBall.

Příloha **E**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src` ..... zdrojové kódy aplikace
- `resources` ..... pomocné soubory aplikace
  - `database_structure.sql` ..... skript pro import databáze
- `docs` ..... dokumentace zdrojového kódu
- `thesis` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `src` ..... zdrojová forma práce ve formátu LaTeX