Ekonomická fakulta
Faculty of Economics

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích
Ekonomická fakulta
Katedra aplikované matematiky a informatiky

Diplomová práce

Vývoj webového ekonomicko-právního informačního systému

Vypracoval: Bc. Jan Pech
Vedoucí práce: Mgr. Radim Remeš

České Budějovice 2019

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
Ekonomická fakulta
Akademický rok: **2018/2019**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Jan PECH**

Osobní číslo: **E17517**

Studijní program: **N6209 Systémové inženýrství a informatika**

Studijní obor: **Ekonomická informatika**

Název tématu: **Vývoj webového ekonomicko-právního informačního systému**

Zadávající katedra: **Katedra aplikované matematiky a informatiky**

Zásady pro vypracování:

Cílem práce je vytvořit webovou aplikaci sloužící jako informační systém pro správu ekonomicko-právních dat. Aplikace bude tvořena několika moduly, např. správa uživatelů, včetně přidělování rolí a oprávnění jednotlivým uživatelům, správa licencí, autentizace a správa unikátnosti přístupu uživatelů, správa obsahu, vyhledávání a usnadnění přístupu k datům, včetně použití filtrů při vyhledávání, platební modul, aj.

Metodický postup:

1. Studium odborné literatury.
2. Návrh, popis vývoje a implementace aplikace a použitých modulů.
3. Zhodnocení, vypracování doporučení a závěrů.

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **50 - 60 stran**

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

1. **Armand, S. (2014).** *Extending Symfony 2 Web Application Framework.* **Birmingham, UK: Packt.**
2. **Duckett, J. (2014).** *JavaScript and JQuery: Interactive Front-End Web Development.* **Chichester, West Sussex, UK: Wiley & Sons.**
3. **CHaffer, J., Swedberg, K. (2013).** *Mistrovství v jQuery: Kompletní průvodce vývojáře.* **Brno: Cpress.**
4. **Kroenke, D. M., Auer D. J. (2014).** *Database Concepts.* **7. vydání. Upper Saddle River, NJ, USA: Prentice Hall.**
5. **Lecky-Thomson, E., Nowicki, S. D. (2010).** *PHP 6: Programujeme profesionálně.* **Brno: Cpress.**
6. **Nixon, R. (2014).** *Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5.* **4. vydání. Sebastopol, CA: O'Reilly.**
7. **Salehi, S. (2016).** *Matering Symfony.* **Birmingham, UK: Packt.**
8. **Wazlawick, R. S. (2014).** *Object-Oriented Analysis and Design for Information Systems.* **Waltham, MA, USA: Morgan Kaufmann.**

Vedoucí diplomové práce: **Mgr. Radim Remeš**

Katedra aplikované matematiky a informatiky

Datum zadání diplomové práce: **26. října 2018**

Termín odevzdání diplomové práce: **12. dubna 2019**

doc. Ing. Ladislav Rolínek, Ph.D.
děkan

L.S.

JIHOČESKÁ UNIVERZITA
V ČESKÝCH BUDĚJOVICÍCH
EKONOMICKÁ FAKULTA
Studentská 13 (1)
370 05 České Budějovice

doc. RNDr. Jana Klicnarová, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 26. října 2018

Prohlášení

Prohlašuji, že svou diplomovou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury. Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své diplomové práce, a to – v nezkrácené podobě/v úpravě vzniklé vypuštěním vyznačených částí archivovaných Ekonomickou fakultou – elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

Datum	Podpis studenta

Poděkování

Nejprve bych chtěl poděkovat vedoucímu mé diplomové práce Mgr. Radimu Remešovi. Vždy, když bylo potřeba, mi vstřícně a ochotně poskytl pomocnou ruku. Navíc měl věcné poznámky a připomínky, které velmi napomohly k úspěšnému dokončení této diplomové práce. Další, komu musím poděkovat, jsou má rodina a přátelé, kteří mi byli oporou po celou dobu mého navazujícího studia a pomáhali mi překonat těžké chvíle, které se během této doby objevily.

Obsah

1 Úvod

Právní předpisy České republiky ovlivňují činnost všech právních a fyzických osob, a pro velmi mnoho z nich je důležité, aby k těmto informacím měly snadný přístup. K zajištění přístupu mohou využít různé způsoby (tištěné Sbírky zákonů a zákoníků, konzultace s právníkem, informační systémy atd.). Nejpopulárnější a pro uživatele nejpřívětivější jsou informační systémy, které oproti ostatním médiím umožňují mnoho funkcionalit navíc (rychlé vyhledávání v právních předpisech, zobrazení historických znění právních předpisů, psaní poznámek, zobrazení vazeb mezi právními předpisy atd.). Na trhu s informačními systémy, které nabízejí právní a ekonomické informace, se pohybuje mnoho firem.

Jednou z těchto firem je i společnost GRAND s.r.o., která byla založena v roce 1997 a přes dvě desetiletí se pohybuje v tomto odvětví. Hlavní činností této firmy je zpracování různých zdrojů dat (Sbírky zákonů, věstníky Ministerstev České republiky, vyhlášky měst a krajů atd.), která využívá interně vyvinutý ekonomicko-právní informační systém EPIS®, jenž je následně prodáván obchodními zástupci firmy koncovým uživatelům. EPIS® má dvě mutace, a to verzi pro desktopové počítače (EPIS® Off-line) a verzi webové aplikace (EPIS® Online).

1.1 Cíl práce

Cílem praktické části této práce je čtenáři přiblížit technologii aplikačního frameworku Symfony 3.4, která je využívána pro vývoj webových aplikací. Praktická část práce se zabývá konkrétním využitím Symfony při vývoji některých komponent systému EPIS® Online. Jelikož je systém EPIS® Online velmi rozsáhlý, jsou v této práci rozebrány pouze jeho hlavní komponenty: modul právních předpisů (datový model, přístup k předpisům, vyhledávání v předpisech, jejich zobrazení), modul uživatelů a jejich licencí (datový model, autentizace, zajištění unikátnosti přístupů – na účet může být v jednu chvíli přihlášení jen jediný uživatel) a modul platební brány sloužící k nákupu licencí. Systém EPIS® Online je možno nalézt na adrese https://novyepis.cz.

2 Webové aplikace

Webové aplikace jsou druhem aplikací, které využívají klient-server architekturu a uživatel se serverem komunikuje pomocí webového prohlížeče. Tento druh aplikací získává na popularitě a pomalu nahrazuje desktopové aplikace[1]. Lze vypozorovat trend, že u mnoha desktopových aplikací vznikají jejich webové mutace (Microsoft Office 365, Skype apod.), které díky moderním technologiím webových prohlížečů poskytují ty samé funkcionality, co jejich desktopové verze. Důvodů, proč k tomuto dochází, je několik (Elizabeth, 2019):

- Není potřeba nic stahovat a instalovat. Uživatelům stačí navštívit adresu webové aplikace a hned ji mohou používat. Nemusí také aplikaci aktualizovat.
- Provozovatel aplikace má jednoduchý přístup k různorodým analytickým údajům (na jaké prvky webové stránky uživatel kliká, na jaké prvky webové stránky uživatel najel myší a jak dlouho nad nimi s kursorem setrval apod.)
- Vývoj webových aplikací je většinou levnější než vývoj desktopových aplikací. Často je požadováno, aby desktopové aplikace běžely na různých platformách (Windows, iOS, Linux apod.) a tak se musí vyvíjet několik verzí aplikace uzpůsobených na cílovou platformu. U webové aplikace stačí vyvinout jednu její verzi uzpůsobenou na webový server, na kterém poběží.

Existují však aplikace, pro které volba webové aplikace není nejvhodnější, jelikož jejich ovládání prostřednictvím webového prohlížeče je pro uživatele obtížné, potřebují velký výpočetní výkon apod. (Elizabeth, 2019)

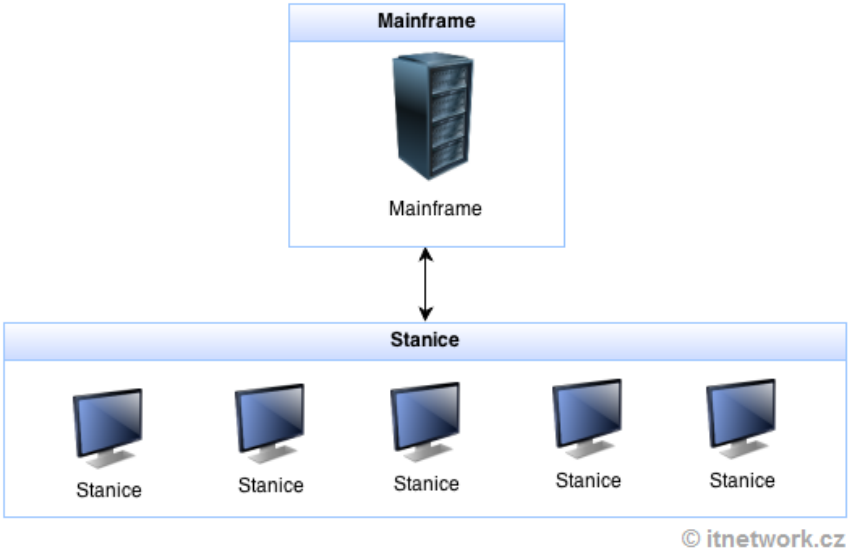
U moderních webových aplikací je kladen důraz na responzivní design uživatelského rozhraní. To znamená, že se aplikace vhodně přizpůsobí na menší obrazovku mobilních zařízení. Ač se to většině webových aplikací díky moderním nástrojům (Bootsrap, jQuery atd) daří, většina uživatelů mobilních zařízení raději využívá mutaci aplikace určené pro jejich zařízení, než aby aplikaci používali z webového prohlížeče jejich přístroje.

[1] Desktopové aplikace jsou takové aplikace, které běží na počítači, na kterém byli nainstalovány. Jsou spouštěny z pevného disku počítače a využívají jeho operační paměť pro své fungování. (Elmbland, 2019)

2.1 Historie webových aplikací

V počátcích informační éry (50. léta až 80. léta 20. století) byly počítače velmi drahé a nebylo možné, aby každý zaměstnanec ve firmě měl svůj vlastní počítače. Proto se zakoupil jeden centrální (tzv. mainframe) počítač a k tomuto centrálnímu počítači bylo připojeno několik klávesnic a monitorů, které využívali zaměstnanci firmy. Schéma tohoto modelu je k na obrázku 1 (Čápka, 2019).

Obrázek 1 – Princip mainframe

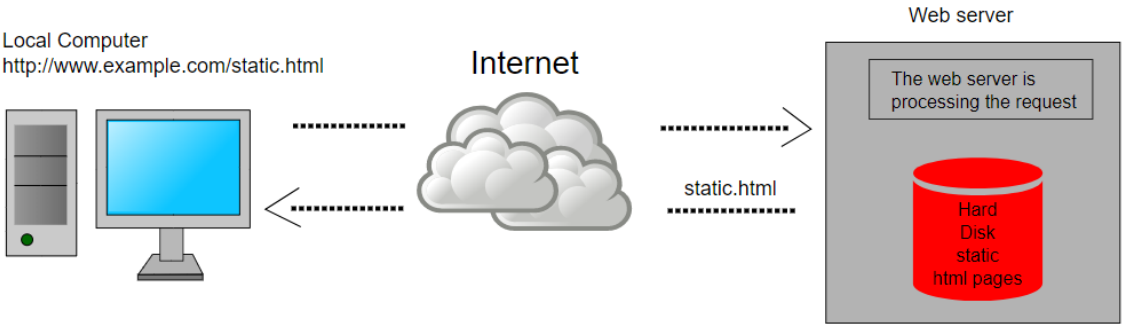


(Zdroj: https://www.itnetwork.cz/images/5/php/mainframe.png)

S technologickým rozvojem se snižovala cena počítačů a mohli si je dovolit menší firmy i domácnosti, a proto nebyl důvod, aby každý zaměstnanec neměl svůj počítač. Výhody desktopových aplikací jsou vysoký výkon (závisí na počítači, na kterém je aplikace nainstalována) a jejich nevýhody jsou složitá správa (instalace, aktualizace apod.) a nízká bezpečnost (na PC je k dispozici celá aplikace, z které lze získat zdrojový kód – dissasblemovat) (Čápka, 2019).

Dalším krokem (konec 20. století), který vedl ke vzniku webových aplikací, byly statické webové stránky, které se tvoří pomocí značkovacího jazyka HTML. Při dotázání adresy byla uživateli do prohlížeče serverem poslána webová stránka, která se neměnila (bylo ji možné jen číst). Této architektuře se říká klient-server (Obrázek 2). Výhody tohoto přístupu jsou malá zátěž (server zasílá pouze HTML stránky a nezajímá se o jejích zobrazení – to probíhá v prohlížeči uživatele), snadná správa (jakmile je na webovém serveru provedena změna, tak se promítne všem uživatelům) a vysoká bezpečnost (klient nemá k dispozici zdrojový kód aplikace) (Čápka, 2019).

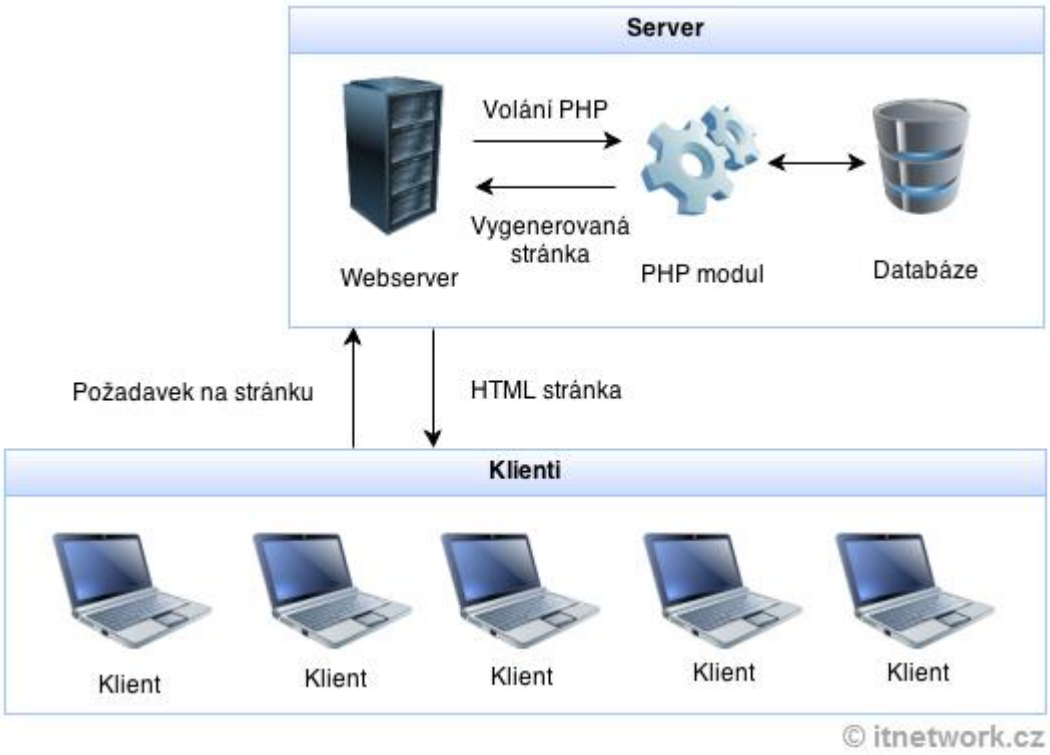
Obrázek 2 - Princip webových stránek



(Zdroj: https://upload.wikimedia.org/wikipedia/commons/5/57/Scheme_static_page_en.svg)

Po velkém rozmachu internetu a webových stránek se hledal způsob, jak do těchto stránek přidat dynamické funkcionality. To vedlo k zavádění nových funkcionalit do HTML značkovacího jazyka a vzniku skriptovacího jazyka JavaScript, který umožňuje měnit statické prvky webových stránek. Dále se rozvíjely programovací jazyky běžící na webových serverech, které umožňují serveru reagovat na specifické požadavky uživatelů. Tento pokrok umožnil to, že se statické webové stránky změnily na dynamické (jejich obsah se mění v závislosti na tom, co klient provádí a požaduje) a tyto stránky se chovají stejně jako desktopové aplikace. Těmto webovým stránkám se říká webová aplikace (pořád se jedná o klient-server architekturu - Obrázek 3). Webové aplikace mají všechny výhody statických webových stránek (Čápka, 2019).

Obrázek 3 – Princip webových aplikací



(Zdroj: https://www.itnetwork.cz/images/5/php/php_webove_aplikace.png)

2.2 Frameworky pro vývoj webových aplikací

Framework (nebo také softwarový framework) je soubor různých řešení pomáhající při vývoji aplikace. Poskytuje základy, které vývojář může využít k vyvinutí vlastního programu. Pod těmito základy si lze představit funkcionality, které musí řešit každá aplikace a je zbytečné, aby je vývojář řešil sám. Při použití frameworku se může oprostit od řešení již vyřešených problémů a může se plně soustředit na vývoj

specifických funkcionalit dané aplikace. Framework určený pro webové aplikace může například obsahovat řešení pro směrování požadavků od uživatelů na server a vývojář tudíž programuje jen funkcionality řešící tyto požadavky a nemusí řešit směrování, které může být v některých případech velmi složité (Christensson, 2019).

Frameworků určených pro vývoj webových aplikací je velký počet. Většina z nich je postavena na Model-View-Controller (zkratka MVC, více v MVC softwarová architektura) architektuře a poskytuje řešení pro přístup k databázi, tvorbu šablon HTML stránek, caching[2], autorizaci uživatelů, scaffolding[3] apod. ("Web framework", 2019)

Mezi nejpoužívanější webové frameworky patří ASP.NET, Ruby On Rails, Node.js, Laravel, Django a Symfony ("Technologies market share", 2019). Je zajímavé, že každý z těchto frameworků využívá jiný programovací jazyk (až na Laravel a Symfony, které oba používají PHP, jelikož je Laravel založen na komponentách Symfony), takže nelze říci, že určitý programovací jazyk je pro vývoj webových aplikací nejvhodnější.

2.3 MVC softwarová architektura

MVC je zkratka pro Model-View-Controller, což jsou 3 základní části, z kterých je tato architektura sestavena.

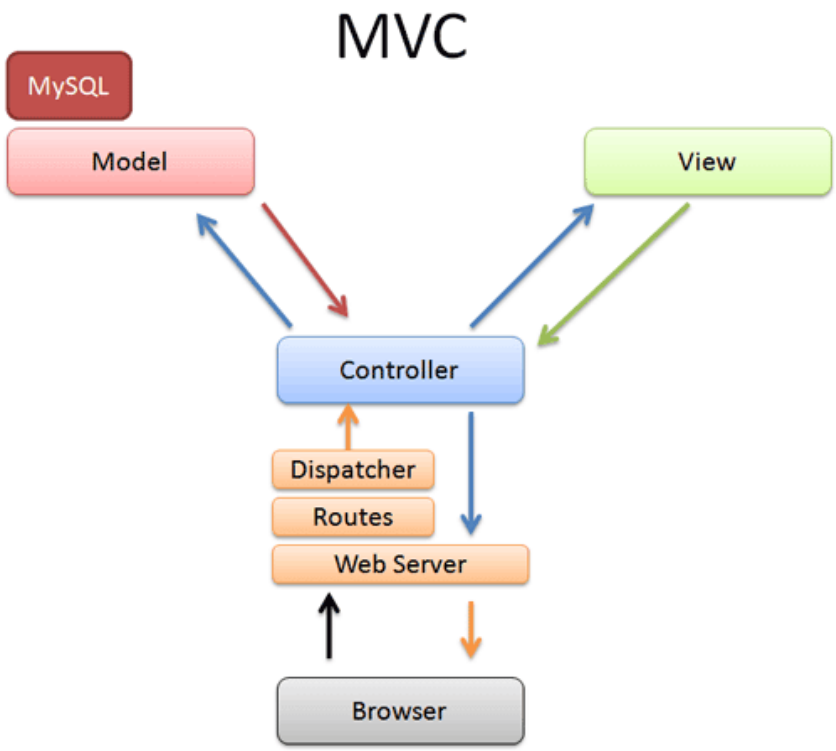
- Model (česky model) – Zde si lze představit datovou část aplikace. Např. uživatel, či článek, které aplikace načetla z databáze a model je jejich abstraktivním znázorněním v programovacím jazyce.
- View (česky pohled) – Tato část je sestavená grafická vizualizace odpovědi serveru na dotaz uživatele (např. HTML stránka).
- Controller (česky řadič) – Tato složka MVC architektury všechno spojuje a dává dohromady. Od uživatele získá nějaký požadavek, ten zpracuje k čemuž využije informace z modelů aplikace a na základě výsledku této operace sestaví pohled, který vrátí uživateli.

Schéma MVC architektury u webové aplikace je znázorněno na obrázku 4. Následuje příklad, jak postupuje webový e-shop, když uživatel klikne na nákupní košík. Webový

[2] Caching je přístup, kdy výsledky často prováděných operací, které jsou náročné na provedení, se někam uloží a když jsou potřeba, tak se nahrají z úložiště.
[3] Scaffolding je výraz pro technologii, která umožňuje na základě několika parametrů vygenerovat zdrojový kód pro nějaký problém (např. formulář pro vytvoření uživatele).

server zaznamená požadavek a pošle ho ke zpracování vrstvě, která se stará o směrování požadavků. Ta z URL adresy požadavku pozná, že uživatel chce zobrazit košík a nasměruje požadavek na správný řadič. Řadič starající se o zobrazení nákupního košíku načte model uživatele a obsah jeho košíku. Tyto informace předá pohledu, který na základě těchto dat vygeneruje HTML stránku zobrazující košík uživatele.

Obrázek 4 – MVC architektura u webové aplikace



(Zdroj: http://1.bp.blogspot.com/_R2pbFBgV4uk/TItwslMxZPI/AAAAAAAAA9s/vu80e5mAbEY/s1600/mvc-rails.png)

2.4 Symfony

Symfony je framework určený k vývoji webových aplikací, využívá MVC architekturu a momentálně je velmi populární. V současné době na něm pracuje přes 3000 vývojářů (přispěvatelů), kteří ho rozvíjí dál. První verze Symfony vyšla v roce 2005 pod MIT Open Source licencí. Symfony je známé svou rozsáhlou komunitou, která čítá přes 600 000 členů. Díky své popularitě existuje velké množství připravených řešení, která každý může využít v Symfony projektu (např. řešení umožňující autentizaci uživatelů – autentizace uživatele, funkcionalita zapomenutého hesla, profil uživatele).

Tuto popularitu dokazuje i to, že v současné době má okolo 48 milionů stažení za měsíc ("Symfony", 2019).

Symfony je tvořeno mnoha na sobě nezávislými komponentami. Vývojář tudíž nemusí použít všechny komponenty, ale může si vybrat jen ty, které se mu hodí. Toho využívají další frameworky (např. Laravel, Drupal, phpBB, eZ Publish atd.), které jsou postaveny na některých komponentách Symfony (Salehi, 2016).

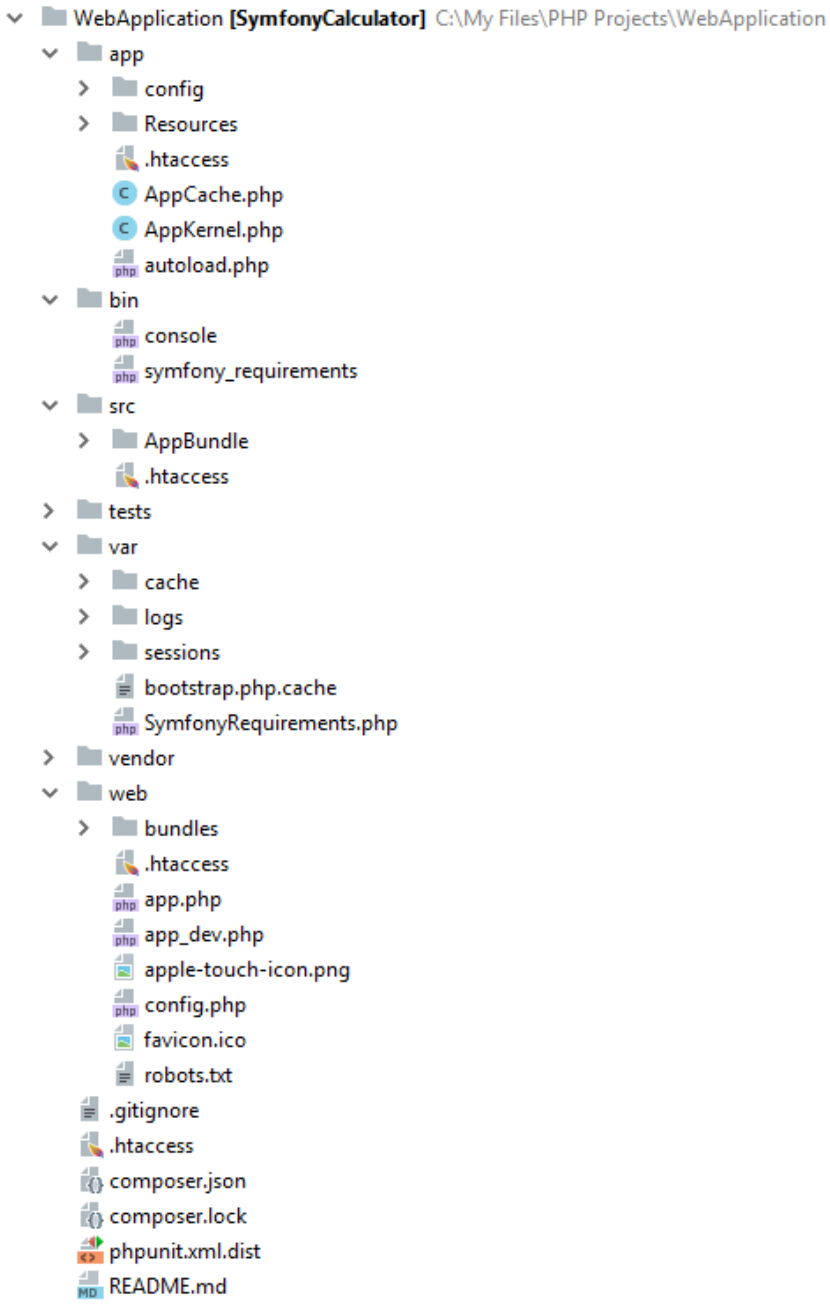
Salehi říká, že Symfony je jedním z nejlepších frameworků pro vývoj webů. Nabízí solidní výsledky při benchmarkových testech, je jednoduché na údržbu, jeho architektura umožňuje lehkou upravitelnost projektu a nabízí vývojářům strmou učící křivku. Symfony užívají velké společnosti jako BBC nebo CBS, jejichž stránky zaznamenávají velké zatížení a Symfony se jim osvědčilo (Salehi, 2016).

V podkapitolách kapitoly Symfony jsou popsány komponenty, které byly využity při vývoji modulů EPIS® Online vyjmenovaných v cílech této práce (kapitola Cíl práce). EPIS® Online byl vyvinut pomocí Symfony verze 3.4., tudíž jsou zmíněné komponenty popsány ve stavu, ve kterém se nacházejí v této verzi Symfony. Jelikož je Symfony živý neustále se vyvíjející systém, tak je možné a dosti pravděpodobné, že některé komponenty se v předchozích či následujících verzích Symfony chovají jinak.

2.4.1 Struktura Symfony projektu

Struktura Symfony projektu je velmi jednoduchá a přehledná. Kořenový adresář projektu obsahuje složky: app, bin, src, tests, var, vendor a web (Obrázek 5). Dále se v něm nachází soubory technologie Composer, která je popsána v kapitole Správa přídavných balíčků a může (zaleží na konfiguraci webového serveru) obsahovat .htaccess soubor (Armand, 2014).

Obrázek 5 – Struktura Symfony projektu



(Zdroj: Autor)

Složka app, obsahuje složky config a Resources. V složce config se nacházejí konfigurační soubory, které jsou dále popsány v kapitole Konfigurační soubory. Ve složce Resources se nalézá složka public, která se dále dělí na složky js (tato složka obsahuje JavaScriptové soubory) a css (tato složka obsahuje kaskádové styly aplikace). Dále se v složce Resources nachází složka views, která obsahuje soubory šablon s příponou .twig, které jsou dále popsány v kapitole Šablonovací systém Twig. Dále je zde umístěn soubor AppKernel.php, který slouží k zapínání a vypínání přídavných balíčků (více v kapitole Správa přídavných balíčků).

V složce bin se nachází důležitý soubor console. Ten se využívá k exekuci příkazů, které Symfony obsahuje. Pokud se v příkazový řádek nachází v kořenovém adresáři projektu, zadáním příkazu zobrazeným na obrázku 6 se vypíší všechny dostupné příkazy. Existují příkazy na generování formulářů, čištění cache a mnoho dalšího. Doinstalované balíčky mohou obsahovat další příkazy a vývojář si může dokonce nadefinovat příkazy vlastní.

Obrázek 6 – Symfony příkaz na vypsání všech dalších příkazů

(Zdroj: https://symfony.com/doc/3.3/console/usage.html)

Složka tests obsahuje místo, kam vývojář ukládá unit testy na jednotlivé komponenty aplikace. Ve složce var je umístěna složka cache (zde se ukládají soubory cache) a složka logs, kam se ukládají logy (výjimky, vývojářem definované logy apod.). Ve složce vendor se nacházejí jednotlivé doinstalované balíčky důležité pro chod aplikace.

Složka web je vstupním bodem aplikace. Do této složky by měly být serverem směřovány veškeré požadavky. Nachází se zde .htaccess soubor, který požadavky směřuje buď na soubor app.php, nebo app_dev.php. Symfony defaultně obsahuje dva režimy, a to režim prod (produkční režim) a režim dev (vývojářský režim). Pokud se směřují požadavky na app.php, je zapnutý produkční režim, který je určen k použití při nasazení aplikace. Vývojářský režim se zapne směrováním požadavků na app_dev.php. Tento režim slouží k odstraňování chyb a rychlejší vývoj, jelikož poskytuje při pádech aplikace a vzniklých výjimkách daleko více informací. Do této

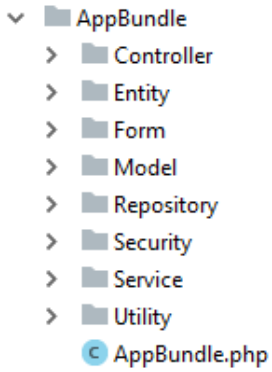
složky se také ukládají soubory určené k přístupu uživateli (obrázky na webu, příkazem se sem kopírují JavaScriptové soubory a soubory kaskádových stylů ze složky Resources popsané výše apod.).

Složka src obsahuje zdrojový kód aplikace. Většinou se zde nachází složka, jejíž název se tvoří názvem aplikace a sufixem Bundle (balíček) (Obrázek 7). Aplikaci se může rozdělit do několika balíčků podle toho, k čemu slouží (např. ShopBundle – část aplikace řešící e-shop, GalleryBundle – část aplikace řešící zobrazení správu obrázků apod.). Vytvořený balíček, pokud splňuje určité předpoklady, je možné nabídnout dalším vývojářům k dispozici a ti ho mohou stáhnout a využít ve své aplikaci.

Samotný balíček aplikace se dělí na několik složek dle toho, co se do nich podle logického významu ukládá:

- Složka Controller – Zde se ukládají řadiče (controllers) aplikace.
- Složky Entity, Model a Repository – Zde se ukládají soubory pracující s datovou stránkou aplikace.
- Složka Form – Zde se ukládají třídy umožňující generování formulářů.
- Složka Security – Zde se ukládají bezpečnostní prvky aplikace.
- Složka Service – Zde se ukládají třídy obsahující logiku aplikace (např. třída zajišťující komunikaci s fakturačním systémem).
- Složka Utility – Zde se ukládají pomocné třídy (např. třída obsahující pomocné metody pro práci s textovými řetězci).

Obrázek 7 – Struktura balíčku aplikace



(Zdroj: Autor)

2.4.2 Správa přídavných balíčků

Symfony využívá k správě přidaných balíčků technologii Composer. Ta umožňuje definovat, na kterých knihovnách je aplikace závislá a následně je dovoluje pomocí příkazů nainstalovat či aktualizovat. Tato definice se provádí v souboru composer.json v kořenovém adresáři aplikace. V tomto souboru se nachází oddíle require (Obrázek 8), kde se definují knihovny, které aplikace potřebuje.

Obrázek 8 – Oddíl require v souboru composer.json

```
"require": {
    "php": ">=5.5.9",
    "doctrine/doctrine-bundle": "^1.6",
    "doctrine/doctrine-cache-bundle": "^1.2",
    "doctrine/orm": "^2.5",
    "incenteev/composer-parameter-handler": "^2.0",
    "sensio/distribution-bundle": "^5.0",
    "sensio/framework-extra-bundle": "^3.0.2",
    "symfony/monolog-bundle": "^3.0.2",
    "symfony/polyfill-apcu": "^1.0",
    "symfony/swiftmailer-bundle": "^2.3.10",
    "symfony/symfony": "3.2.*",
    "twig/twig": "^1.0||^2.0"
},
```

(Zdroj: Autor)

Pokud je doinstalovaná knihovna Symfony balíček, tak je ještě nutné ho aktivovat v souboru AppKernel.php. V tomto souboru se nachází třída (Ukázka PHP kódu 1), která obsahuje metodu registerBundles. V této metodě se musí do proměnné $bundles přidat instanci doinstalovaného Symfony balíčku.

Ukázka PHP kódu 1 – Soubor AppKernel.php

```php
class AppKernel extends Kernel
{
    public function registerBundles()
    {
        $bundles = [
            new Symfony\Bundle\FrameworkBundle\FrameworkBundle(),
            new Symfony\Bundle\SecurityBundle\SecurityBundle(),
            new Symfony\Bundle\TwigBundle\TwigBundle(),
            new Symfony\Bundle\MonologBundle\MonologBundle(),
            new Symfony\Bundle\SwiftmailerBundle\SwiftmailerBundle(),
            new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
            new Sensio\Bundle\FrameworkExtraBundle\SensioFrameworkExtraBundle(),
            new CalculatorBundle\CalculatorBundle()
        ];

        if (in_array($this->getEnvironment(), ['dev', 'test'], strict: true)) {
            $bundles[] = new Symfony\Bundle\DebugBundle\DebugBundle();
            $bundles[] = new Symfony\Bundle\WebProfilerBundle\WebProfilerBundle();
            $bundles[] = new Sensio\Bundle\DistributionBundle\SensioDistributionBundle();
            $bundles[] = new Sensio\Bundle\GeneratorBundle\SensioGeneratorBundle();
        }

        return $bundles;
    }
```

(Zdroj: Autor)

2.4.3 Konfigurační soubory

Ve složce config se nacházejí konfigurační soubory projektu. Těmito hlavními soubory jsou config.yml, parameters.yml, services.yml a security.yml. Soubory security.yml a services.yml jsou popsány dále v této práci (kapitoly Zabezpečení aplikace a Služby)

Soubor config.yml obsahuje nastavení jednotlivých balíčků, z kterých se Symfony skládá a také se sem zadává nastavení doinstalovaných balíčků Symfony. Každý balíček potřebuje jiné nastavení a struktura konfigurace se různí. Na obrázku 9 je zobrazena konfigurace balíčku Swiftmailer, což je komponenta umožňující rozesílání e-mailů. Má 5 povinných nastavení, která se musí zadat. Zeleným textem v jednoduchých uvozovkách je znázorněna syntaxe, jak se předávají balíčku do hodnot nastavení parametry ze souboru parameters.yml.

Obrázek 9 – Konfigurace balíčku Swiftmailer

```
# Swiftmailer Configuration
swiftmailer:
    transport: '%mailer_transport%'
    host: '%mailer_host%'
    username: '%mailer_user%'
    password: '%mailer_password%'
    spool: { type: memory }
```

(Zdroj: Autor)

Soubor parameters.yml obsahuje parametry aplikace (přístupové údaje k databázím, přístupové údaje k API fakturačního systému apod.), které se pak předají jejím jednotlivým částím (doinstalovaným balíčkům, službám, řadičům, šablonám atd.). Výhoda v tomto přístupu je, že jsou všechny parametry u sebe a při jejich změně se nemusí sahat do zdrojového kódu nějaké části aplikace, ale stačí upravit jen tento konfigurační soubor.

Obrázek 10 – Soubor parameters.yml

```
parameters:
    database_host: 127.0.0.1
    database_port: null
    database_name: symfony
    database_user: root
    database_password: null
    mailer_transport: smtp
    mailer_host: 127.0.0.1
    mailer_user: null
    mailer_password: null
    secret: 6e07e1093a8c149bb4488b4a0d3c2e7139c1cbe1
```

(Zdroj: Autor*)*

2.4.4 Controller

V ukázce PHP kódu 2 je zobrazen zdrojový kód vytvořeného řadiče. U tvorby řadiče je důležité, aby se nacházel ve složce Controller daného balíčku a dědil z komponenty Symfony\Bundle\FrameworkBundle\Controller\Controller. To zajistí, že každá jeho metoda se sufixem Action a anotací Route bude zaznamenána jako cílová stanice požadavku. Metoda indexAction bude zavolaná ve chvíli, kdy uživatel navštíví adresu (např. webová aplikace má adresu https://web-app.com) https://web-app.com/nakupni-kosik. Aby bylo možné tuto cestu generovat v kódu, tak je nutno ještě do anotace uvést parametr name. Metodám s anotací Route a sufixem Action se říká akce.

Ukázka PHP kódu 2 – Ukázka tvorby směrování

```php
namespace AppBundle\Controller;


use AppBundle\Service\PurchaseGate;
use AppBundle\Model\Entity\ShoppingItem;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;

class ShoppingCartController extends Controller
{
    /**
     * @param Request $request
     * @Route("/nakupni-kosik", name="app_shopping_cart_index")
     */
    public function indexAction(Request $request)
    {
        //controller code
    }
}
```

(Zdroj: Autor)

Akce indexAction má parametr $request určitého datového typu. Symfony skrze důmyslný systém nazvaný Depency Injection, zajistí, že za tento parametr bude dosazena instance daného datového typu. Takto je možné do akcí řadičů předávat instance služeb. Některé služby jsou v Symfony předefinované (např. zmíněný parametr $request datového typu Request – tato služba poskytuje informace o současném požadavku na server – o jakou se jedná metodu, jaké byly předány parametry v URL atd.). Samozřejmě si vývojář může nadefinovat vlastní služby. Do jejich konstruktoru se mohou uvést jako parametry další služby a Symfony zajistí, že jsou správně předány (více o Depency Injection v kapitole Služby).

2.4.5 Doctrine, entity, modely a repositáře

Symfony neposkytuje nástroje pro práci s databází, ale poskytuje integraci s knihovnou Doctrine ("Symfony", 2019). Ta byla využita při vývoji EPIS® Online. Doctrine je nástroj zajišťující automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem PHP, tzv. objektově relační mapování ("Objektově relační mapování", 2019).

V ukázce PHP kódu 3 lze vidět, jak se provádí mapování pomocí anotací. Tyto anotace slouží k propojení objektového modelu a databáze. Těmto třídám se v Symfony říká entity a ukládají se do složky Entity. Anotace Table umožňuje zadat název tabulky databáze, do které budou ukládána data třídy. Anotace Entity slouží k propojení s repositářem, který je vysvětlen dále v této kapitole. Důležitá anotace je Column nad datovými atributy. Ta umožňuje zadat parametry sloupce v tabulce (název sloupce, datový typ sloupce, unikátnost hodnost ve sloupci atd.). Nad datovým atributem $id jsou další anotace, které znázorňují, že se jedná o jedinečný identifikátor a jaká má být zvolena strategie pro generování tohoto identifikátoru.

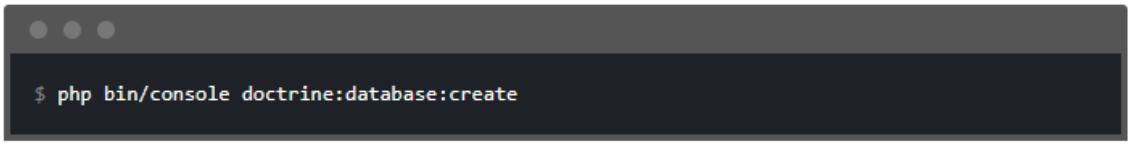
Ukázka PHP kódu 3 – Mapování pomocí anotací

```php
namespace AppBundle\Model\Entity;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Table(name="shopping_item")
 * @ORM\Entity(repositoryClass="AppBundle\Utility\ShoppingItemRepository")
 */
class ShoppingItem
{
    /**
     * @var int
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @var string
     * @ORM\Column(name="name", type="string", unique=true)
     */
    private $name;
}
```

(Zdroj: Autor)

Vývojář využívající Symfony nemusí vůbec používat nástroje pro správu databáze. Stačí mu všechny vazby správně zadat v kódu aplikace a pomocí příkazu níže (Obrázek 11) se vytvoří databáze i se schématem, které vytvořil pomocí anotací. Musí však správně zadat přístupové údaje k databázovému serveru v souboru parameters.yml. Doctrine také umožňuje mapovat všechny typy vazeb, které se mohou objevit mezi třidami (1:1, 1:N, N:M).

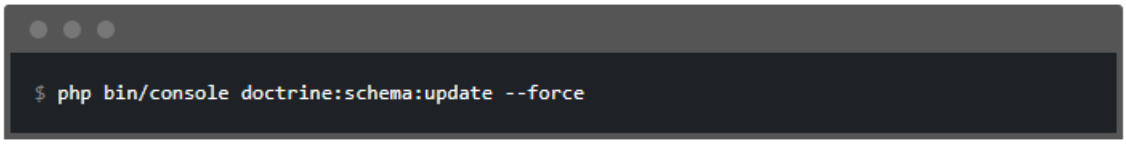
Obrázek 11 – Symfony příkaz pro vytvoření databáze



(Zdroj: https://symfony.com/doc/3.3/doctrine.html)

Pokud programátor má již vytvořenou databázi a potřebuje pouze aktualizovat její schéma (např. změnil vazby mezi některými třídami), tak stačí zadat příkaz níže (Obrázek 12).

Obrázek 12 – Symfony příkaz pro aktualizaci schéma databáze



(Zdroj: https://symfony.com/doc/3.3/doctrine.html)

Model je to samé, co entita, ale modely jsou uloženy ve složce Model a jejich instance se neukládají do databáze, tudíž většinou slouží k nějakému jednorázovému užití a není potřeba si jejich instance zapamatovat.

Další důležitou komponentou v Symfony jsou repositáře umístěné ve složce Repository. Každý repositář patří určité entitě. Repositář slouží k tomu, že obsahuje pomocné v aplikaci často používané dotazy na databázi. V ukázce PHP kódu 4 je zobrazen repositář entity ze začátku této kapitoly (Ukázka PHP kódu 3). Obsahuje metodu findAllOrderByName, v které využívá QueryBuilder. To je komponenta Doctrine umožňující vytváření SQL dotazů pomocí jednoduchého programovacího rozhraní. Toto rozhraní vytvoří SQL dotaz, odešle ho na databázi a výsledek, který databáze vrátí namapuje podle anotací na třídu v kódu. V ukázce je zobrazeno využití tohoto rozhraní k vyhledání všech záznamů v databázi a jejich seřazení podle datového atributu name

sestupně. Výsledek dotazu je namapován na třídu ShoppingItem a vrácen jako pole. Repositář lze podobně jako službu předávat jiným objektům pomocí technologie Depency Injection.

Ukázka PHP kódu 4 – Ukázka repositáře

```php
namespace AppBundle\Utility;


use AppBundle\Model\Entity\ShoppingItem;
use Doctrine\ORM\EntityRepository;

class ShoppingItemRepository extends EntityRepository
{
    /**
     * @return ShoppingItem[]
     */
    public function findAllOrderedByName()
    {
        $queryBuilder = $this->createQueryBuilder( alias: "shopping_item")
            ->orderBy( sort: "shopping_item.name",  order: "ASC");
        $result = $queryBuilder->getQuery()->getResult();

        return $result;
    }
}
```

(Zdroj: Autor)

S entitami souvisí i technologie ParamConverter. Akce řadiče buyItemAction je volána ve chvíli, kdy uživatel navštíví adresu /nakup/1 (místo čísla 1 zde může být cokoliv). Dále Symfony zkoumá parametry této akce. Do parametru $purchaseGate vloží instanci této třídy, protože zjistí, že je to uživatelem definovaná služba (Definice třídy se nachází ve složce Service) (Více o Dependency Injection v kapitole Služby). Dalším parametrem je $shoppingItem. U toho zjistí, že se jedná o entitu, jelikož definice třídy je ve složce Entity. Symfony projde datové atributy této třídy (Ukázka PHP kódu 3) a najde, že datový atribut $id odpovídá sloupci id v databázi. Hodnotu za lomítkem (v příkladu níže je to 1) využije k vyhledání záznamu v databázi a tento záznam namapuje na daný datový typ. Pokud by žádný záznam podle zadaného identifikátoru Symfony nenašlo, tak uživateli vrátí kód 404 se zprávou, že se záznam nepodařilo nalézt.

Ukázka PHP kódu 5 – Ukázka technologie ParamConverter

```php
/**
 * @param PurchaseGate $purchaseGate
 * @param ShoppingItem $shoppingItem
 * @Route("/nakup/{id}", name="ap_shopping_cart_buy_item")
 */
public function buyItemAction(PurchaseGate $purchaseGate, ShoppingItem $shoppingItem)
{
    //controller code
}
```

(Zdroj: Autor).

2.4.6 Služby

Služby se v Symfony dají rozdělit na dva typy. První typ služeb jsou ty, které jsou předdefinované samotným frameworkem. Většinou se jedná o služby, které nějakým způsobem ovlivňují chod aplikace a nemají nic společného s byznys logikou. V Symfony existuje služba pomáhající pracovat s požadavky, které zaslal uživatel. Dále zde existuje služba pro práci s databází (uchovávání entit v databázi atd.) anebo služba, která umožňuje generovat URL směřující na akce jednotlivých řadičů atd. Druhým typem služeb jsou ty, které si nadefinoval vývojář a většinou obsahují byznys logiku aplikace. Pro tyto služby je určené místo ve složce Service.

V příloze 1 je zobrazeno, jak funguje technologie Dependency Injection. Toto schéma vyobrazuje, jakým způsobem jsou předávány parametry ze souboru parameters.yml službám a jakým způsobem je možné předat jednu službu jiné službě jako argument

konstruktoru. Akci řadiče se předává služba jako parametr dané metody, u kterého je nutno uvést daný datový typ a Symfony samo tento argument vyplní instancí požadované služby.

2.4.7 Formuláře

Formuláře se v Symfony nejčastěji vztahují k nějaké entitě (nebo modelu) a dovolují ním uživateli upravit hodnoty v datových atributech dané entity (nebo modelu). V ukázce PHP kódu 6 je zobrazeno, jak se tvoří formulář umožňující upravit vlastnosti entity (nebo modelu). Do složky Form se vytvoří třída, která má většinou název entity, ke které se formulář vztahuje, se sufixem Type. Tato třída dědí ze třídy AbstractType a přepisuje dvě její metody, a to configureOptions a buildForm. Konstruktorem formuláře lze pomocí Dependency Injection získat instance služeb a parametry ze souboru parameters.yml (obdobně jako tomu je u služeb a akcí řadičů).

Ukázka PHP kódu 6 – Tvorba formuláře

```php
namespace AppBundle\Form;


use AppBundle\Model\Entity\ShoppingItem;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class ShoppintItemType extends AbstractType
{
    /**
     * @inheritdoc
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder->add( child: "name",  type: TextType::class, [
            "label" => "Název",
        ]);
    }

    /**
     * @inheritdoc
     */
    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            "data_class" => ShoppingItem::class
        ]);
    }
}
```

(Zdroj: Autor)

Metoda buildForm (Ukázka PHP kódu 6) je volána ve chvíli, kdy je tvořen formulář. V této metodě pomocí FormBuilderInterface se mohou přidávat jednotlivé elementy formuláře. V příkladu je přidáno metodou add objektu $builder políčko umožňující editaci hodnoty entity datového atributu $name (Ukázka PHP kódu 7). Toto políčko je typu TextType, což představuje jednoduchý formulářový textový vstup. Posledním parametrem metody add je pole s konfigurací. V této konfiguraci je možno předat text popisku formulářového políčka, HTML atributy vygenerovaného elementu apod.

Metoda configureOptions (Ukázka PHP kódu 6) slouží k defaultnímu nastavení daného formuláře. Této metoda je předána instance třídy OptionsResolver, která umožňuje konfiguraci formuláře. Metoda setDefaults si bere jako argument pole, v kterém se předávají hodnoty nastavení. V tomto poli lze vypnout CSRF ochranu[4], nastavit, jakou metodu má formulář využít k odeslání dat (POST, nebo GET) atd. Do konfigurace data_class se předává název třídy, ke které se formulář vztahuje.

V ukázce PHP kódu 7 je zobrazeno, jakým způsobem se dají zadat omezující podmínky pro prvky formuláře. Konkrétně je zobrazeno omezení Length z jmenného prostoru Assert. Toto omezení umožňuje zadat minimální a maximální délku hodnotuy kterou uživatel do formuláře musí zadat. Dále je do parametru groups, předáno pole názvů validačních skupin. To se využívá pro odlišení různých validačních pravidel pro různé formuláře. Například je vytvořen jeden formulář, který má v sobě elementy pro správu entity a je žádoucí, aby se formulářem zadané údaje validovaly jiným způsobem při editaci entity a jiným způsobem při vytváření nové entity. Proto se při tvorbě formuláře do nastavení přidává pole názvů validačních skupin a z nastavených omezení se využijí jen ta, která mají danou validační skupinu. Příloha 2 ukazuje, jakým způsobem se nejčastěji využívá formulář. Je zde zobrazena akce řadiče, které jsou předány služby Request a ObjectManager a dále je zde načtena entita ShoppingItem (Ukázka PHP kódu 3) pomocí technologie ParamConverter. Nejdříve se vytvoří instance daného formuláře pomocí metoda createForm (tato metoda vytvoří formulář a zavolá metodu buildForm z ukázky Ukázka PHP kódu 6). Ta si za první argument bere název třídy formuláře, druhým argumentem je daná instance entity, která se bude upravovat hodnotami z formuláře a třetím argumentem je pole konfiguračních hodnot,

[4] Symfony má vestavěnou ochranu vůči CSRF útokům, která se provádí pomocí skrytého formulářového prvku, do kterého je uložena hodnota tokenu. Pokud je odeslán formulář, který nemá validní tento token, tak ho Symfony vyhodnotí jako nevalidní a nedostane se ke zpracování.

které obsahuje názvy validačních skupin. Dále se na nově vytvořené instanci formuláře volá metoda handleRequest, která si bere instanci služby Request, z které čerpá data z požadavku uživatele. Následuje podmínka, jestli byl formulář odeslán, a jestli je dle zadaných omezení validní. Pokud tomu tak není, vrací se vykreslená šablona, zobrazující editační formulář (o šablonách více v kapitole Šablonovací systém Twig). Pokud je však formulář odeslán a validní, proběhne kód uvnitř podmínky. Zde služba ObjectManager pomocí metody persist nastavuje, že se nově zadané hodnoty entity mají uložit do databáze a metodou flush dojde k tomuto uložení a aktualizaci hodnot v databázi. Následuje přidání zprávy, která se uloží do session a je vygenerována uživateli po přesměrování. Posledním krokem v podmínce je přesměrování na jinou stránku.

Ukázka PHP kódu 7 – Tvorba omezení pomocí anotací

```php
namespace AppBundle\Model\Entity;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;

/**
 * @ORM\Table(name="shopping_item")
 * @ORM\Entity(repositoryClass="AppBundle\Utility\ShoppingItemRepository")
 */
class ShoppingItem
{
    /**
     * @var string
     * @ORM\Column(name="name", type="string", unique=true)
     * @Assert\Length(min="1", max="20", groups={"administration_edit"})
     */
    private $name;

    Other code
}
```

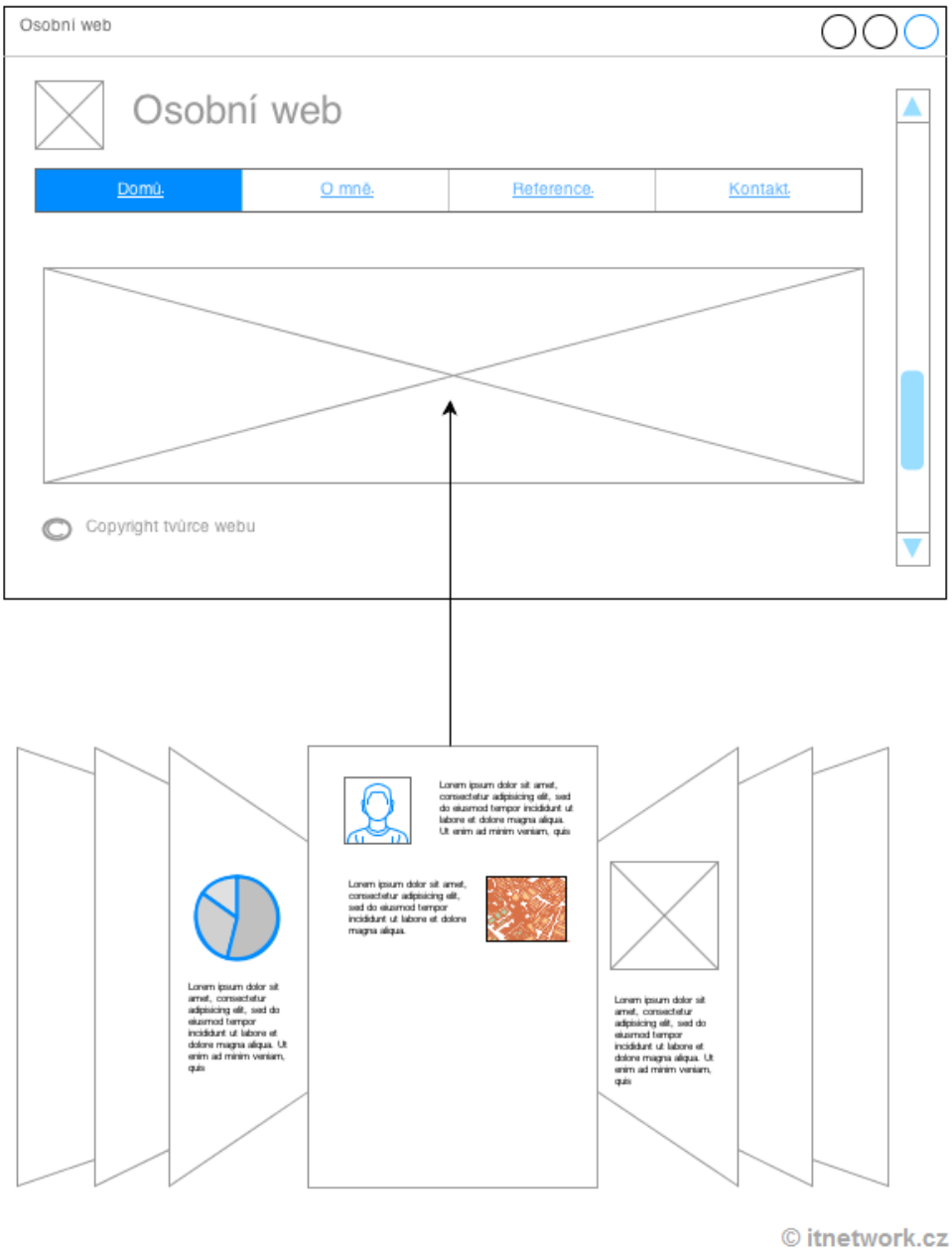
(Zdroj: Autor)

2.4.8 Šablonovací systém Twig

Twig je moderní šablonovací nástroj pro PHP, který byl vytvořen tvůrcem Symfony (Fabien Potencier). Oproti jazyku PHP, který se dá také využít jako šablonovací nástroj, Twig přináší mnoho výhod. Jeho syntaxe je velmi jednoduchá, což šablony vytvořené v tomto jazyku dělá velmi přehledné. Vývojáři poskytuje mnoho zabudovaných funkcionalit (cykly, přístup k atributům instancí objektů, formátování datumu atd.) a dokonce umožňuje vytvořit si funkcionality vlastní ("Twig", 2019).

Twig umožňuje různě skládat jednotlivé šablony do sebe. Většina webových stránek se dá rozdělit na 2 části. První části je hlavní rozložení stránky, které je pro všechny další stránky na webu stejné. Z pravidla se zde nachází hlavička (navigační menu aplikace, její logo atd.) a patička aplikace (copyright atd.). Mezi těmito dvěma částmi je prostor pro obsah jednotlivých stránek. Toto skládání je znázorněno na obrázku 13 (Čápka, 2019).

Obrázek 13 – Skládání šablon do sebe



(Zdroj: https://www.itnetwork.cz/images/5/php/layout_skladani.png)

Příloha 3 zobrazuje jakým způsobem se skládají šablony v Twig a Symfony a jak se předávají proměnné těmto šablonám. V akci řadiče je volána metoda render, která má dva argumenty. Prvním argumentem je textový název šablony, kterou má vykreslit. Druhým argumentem je asociativní pole proměnných, jež se mají předat šabloně. Předané proměnné jsou v šabloně k dispozici pod hodnotami klíčů tohoto asociativního pole. Je zde vidět Twig příkaz extends, který určuje, jakou rodičovskou šablonu má daná šablona rozšířit. Následně může tato šablona využívat bloky definované v rodičovské šabloně. První je zde vyplněný blok title. V tomto bloku je také znázorněno, jakým způsobem se přistupuje v šablonách k datovým atributům předaných proměnných (shopping_item.id) (Ukázka PHP kódu 3). Blok může být v rodičovské šabloně předvyplněn. Pokud není volána speciální funkce parent() (znázorněno v bloku javascripts), je tento předvyplnění obsah nahrazen obsahem z rozvíjející šablony. Pokud je však volána tato funkce, tak se na toto místo dosadí předvyplněný obsah. Dále je v bloku body znázorněno, jakým způsobem dochází k vykreslování jednotlivých polí formuláře (Ukázka PHP kódu 6).

3 Vývoj aplikace

Vývoj webové aplikace EPIS® Online trval necelé dva roky a autor této práce se na ní podílel v týmu s dalším jedním člověkem. Autor práce byl hlavním vývojářem a napsal většinu zdrojového kódu. Data, která využívá EPIS® Online se dají rozdělit na dva druhy. Prvním druhem dat jsou data, která jsou do systému zadávaná zpracovateli dat pomocí interní firemní aplikace pro správu obsahu (dále jen „systém CMS“) (data o právních předpisech a vazby mezi nimi atd.). Druhým druhem dat jsou data, která se vztahují k uživatelům a jejím licencím (uživatelé, licence, aktivace, transakce platební brány atd.). Autor práce navrhl datový model zmíněného druhého druhu dat. Strukturu prvního druhu dat nemohl měnit, jelikož by to znamenalo zásah do systému CMS a musela by se dále provést konverze dat ze staré struktury do nové, což by bylo velmi časově náročné. Jak je patrné z přílohy 4, EPIS® Online první druh dat pouze čte (nemá oprávnění je měnit) a může měnit a upravovat pouze druhý druh dat.

V kapitole Modul uživatelů a jejich licencí je rozebrán datový model uživatelů a jejich licencí, což souvisí i s byznys modelem aplikace, který je zde také rozebrán. Dále je v této kapitole popsáno zabezpečení aplikace, k čemuž je využit balíček FosUserBundle a jakým způsobem bylo zajištěno unikátnosti přístupů (aby jeden účet mohl používat v jeden moment jen jeden uživatel).

Kapitola Modul právních předpisů pojednává o datovém modelu dat právních předpisů, o způsobu vyhledávání v nich, stránkovaní vyhledaných výsledků a o zobrazení konkrétního právního předpisu. V této kapitole je rozebrána Symfony technologie Voter, která umožňuje určit, jestli si uživatel může zobrazit daný právní předpis nebo ne.

V kapitole Modul platební brány se řeší napojení aplikace na online platební bránu GoPay. Toto napojení bylo pro firmu velmi důležité, jelikož předchozí verze EPIS® Online neumožňovala nákup licence přes webovou aplikaci (licence prodávali pouze obchodní zástupci firmy).

3.1 Metodika

EPIS® Online je vytvořený pomocí frameworku Symfony verze 3.4. Aplikace vytvořené v tomto frameworku potřebují ke svému běhu webové servery Apache nebo Nginx. Zdrojový kód je napsaný pomocí PHP verze 7. EPIS® Online využívá

knihovnu Doctrine a její možnosti objektově-relačního mapování k práci s databází (MySQL). Uživatelské rozhraní je vytvořeno pomocí systému Twig a stylování je provedeno pomocí zakoupené webové šablony, která je postavena na technologii Bootstrap 3.3.6 a jQuery 3.3.1. Celý vývoj probíhal pomocí IDE PhpStorm od firmy JetBrains.

3.2 Modul uživatelů a jejich licencí

Pro pochopení datového modelu je důležité si nejdříve přiblížit byznys model celého systému EPIS® Online. Celý systém je rozdělený do několika modulů. Tyto moduly se od sebe odlišují tím, jestli jsou zdarma, nebo uživatel k zobrazení záznamů v tomto modulu musí mít zakoupenou platnou licenci. Všichni uživatelé (přihlášení, nepřihlášení, s platnou licencí atd.) mohou do všech částí systému a nejsou pohybem v něm nějak omezeni. Pokud se však dostanou na stránku zobrazující text, k jehož zobrazení musí mít oprávnění a nemají je, zakryjí se jim texty, jako je znázorněno v příloze 5. Jednotlivé moduly se od sebe odlišují prefixem adresy pro všechny své stránky. Modul právních předpisů má prefix /pravni-predpisy (celá adresa https://novyepis.cz/pravni-predpisy) a ostatní adresy (např. /pravni-predpisy/registrovy-vypis/78724, pravni-predpisy/plne-zneni/54874 atd.) za tímto prefixem spadají do tohoto modulu (více o těchto adresách v kapitole Modul právních předpisů). Většina stránek je přístupná pro uživatele zdarma, ale když si chce zobrazit samotný text právního předpisu, musí mít zakoupenou licenci, jinak se mu zobrazí text se skrytým obsahem.

Dále v systému figurují uživatelé. Tyto uživatelé si mohou zakoupit licenci do systému pomocí platební brány nebo přes obchodního zástupce firmy. Uživatelé mají vazbu na licence, které si zakupují, a na aktivace licencí.

Každá licence je propojena na modul (či moduly), který k ní byl zakoupen, má datum platnosti od a do a počet aktivací, které se k ní mohou vytvořit. Licence může mít jednu nebo více aktivací. Aktivace licence je to, dle čeho se rozhodne, že má uživatel právo zobrazit si text záznamu v placeném modulu (dle ní se rozhoduje, jestli se uživateli zobrazí plný, nebo skrytý text – ne dle vlastnictví licence). Aktivace licence je vázaná na uživatele, který touto aktivací disponuje a licenci, ke které se vztahuje. Jednotlivé aktivace licence spravuje správce licence. Ten může tvořit aktivace a mazat je. V praxi se tedy může stát, že si uživatel 1 zakoupí licenci a svou aktivaci smaže a vytvoří jí

pro uživatele 2, který žádnou licenci nevlastní. Uživatel 1 tím pádem ztratí přístup k textům záznamů modulů, ke kterým si zakoupil licenci (budou se mu zobrazovat zakryté texty), ale uživatel 2 k jejich textům bude mít přístup (má aktivaci k licenci, která umožňuje jejich zobrazení). Spravovat aktivace může stále jen uživatel 1, jelikož je správce licence.

V systému EPIS® Online existuje 10 zpoplatněných modulů: Právní předpisy ČR (tento modul také zahrnuje Právnický slovník, Termínový kalendář, Právní lhůty a Sazby), Finanční zpravodaj MF ČR, Judikatura, Právní předpisy EU, Sněmovní tisky, Věstníky, Vyhlášky měst a krajů, Vzory smluv, Formuláře a tiskopisy a Metodické pokyny ministerstev. Moduly mohou být sdruženy do předvolených balíčků, které jsou k dispozici za zvýhodněnou cenu (např. předvolený balíček EPIS ® Základní systém obsahuje modul Právní předpisy ČR). Z modulů si uživatel může navolit svůj vlastní mix, který nejlépe odpovídá jeho potřebám, nebo si může vybrat jeden z předvolených balíčků.

V tabulkách 1 a 2 jsou zobrazeny aktuální ceny modulů a předvolených balíčků. Cena je uvedena za 1 přístup na 365 dní. Výpočet ceny licence probíhá interním firemním algoritmem, který závisí na:

- na modulech, ke kterým se vztahuje (případně na ceně předvoleného balíčku)
- na počtu přístupů (aktivací)
- na délce období, na kterou se licence vztahuje

Tabulka 1 – Ceník modulů

Název modulu	**Cena**
Právní předpisy ČR	3 500,- Kč
Finanční zpravodaj MF ČR	500,- Kč
Judikatura	3 500,- Kč
Právní předpisy EU	2 000,- Kč
Sněmovní tisky	800,- Kč
Věstníky	500,- Kč
Vyhlášky měst a krajů	1 000,- Kč
Vzory smluv	1 000,- Kč
Formuláře a tiskopisy	800,- Kč
Metodické pokyny ministerstev	150,- Kč

(Zdroj: Autor)

Tabulka 2 – Ceník předvolených balíčků

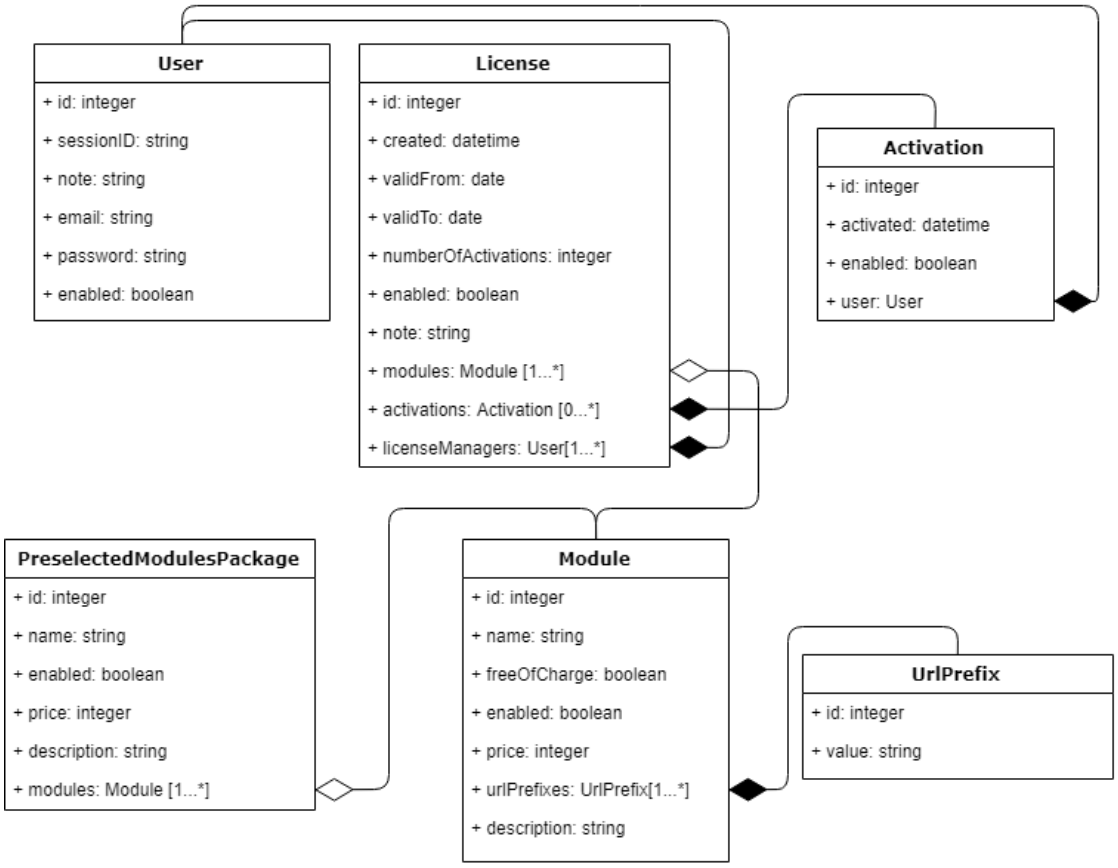
Název předvoleného balíčku	**Cena**
EPIS® Online – Kompletní systém	8 773,- Kč
EPIS® Online – Advokátní praxe	7 840,- Kč
EPIS® Online – Státní a místní správa	6 755,- Kč
EPIS® Online – Základní systém	3 500,- Kč

(Zdroj: Autor)

3.2.1 Datový model

Na obrázku 14 je zobrazen UML diagram tříd modulu uživatelů a jejich licencí. Toto schéma tříd je následně pomocí Doctrine namapováno na tabulky v relační databázi.

Obrázek 14 – Diagram tříd modulu uživatelů a jejich licencí



(Zdroj: Autor)

Třída User reprezentuje uživatele systému. Tato třída má vazbu na licence a aktivace. U této třídy se zaznamenává:

- id – Tento atribut je jedinečný identifikátor instance objektu.

- sessionID – Tento atribut slouží k zajištění unikátnosti přístupů na účet (více v kapitole Zajištění unikátnosti přístupů).
- note – Tento atribut slouží k uchování textové poznámky k uživateli a má interní firemní využití.
- email – Atribut email uchovává e-mailovou adresu uživatele, která je zároveň jeho přihlašovacím jménem.
- password – V tomto atributu je uchováno zašifrované heslo uživatele.
- enabled – Tento atribut slouží k ověření toho, že daná instance objektu je pro program aktivní (může ji využít), nebo není (zachází se s ní jako, kdyby neexistovala).

Třída License reprezentuje licenci v systému a má vazbu na uživatele (respektive správce licence), aktivace a moduly. Popis atributů této třídy:

- id – Tento atribut je jedinečný identifikátor instance objektu.
- created – Atribut created uchovává datum vytvoření licence.
- validFrom – Tento atribut zachycuje hodnotu, od kdy je licence platná.
- validTo – V tomto atributu se ukládá datum, do kdy je licence platná.
- numberOfActivations – Zde se ukládá hodnota, kolik aktivací se může k licenci vytvořit.
- enabled – Tento atribut slouží k ověření toho, že daná instance objektu je pro program aktivní (může ji využít), nebo není (zachází se s ní jako, kdyby neexistovala).
- note – Tento atribut slouží k uchování textové poznámky k licenci a má interní firemní využití.
- modules – Toto je kolekce sdružující moduly, ke kterým se licence vztahuje.
- activations – Tato kolekce uchovává aktivace licence.
- licenseManagers – V této kolekci jsou zachyceni správci licence.

Třída Activations modeluje aktivaci a má vazbu na uživatele a licenci. Obsahuje atributy:

- id – Tento atribut je jedinečný identifikátor instance objektu.
- activated – Zde je uchováno datum, kdy byla aktivace vytvořena.
- enabled – Tento atribut slouží k ověření toho, že daná instance objektu je pro program aktivní (může ji využít), nebo není (zachází se s ní jako, kdyby neexistovala).

- user – Zde je uložena instance uživatele, ke kterému se aktivace vztahuje.

Třída Module znázorňuje modul v systému a má vazbu na předvolený balíček, licenci a prefix adresy. V této třídě se nacházejí atributy:

- id – Tento atribut je jedinečný identifikátor instance objektu.
- name – Zde je uložen název modulu.
- freeOfCharge – Tato hodnota zachycuje, jestli je modul zdarma, nebo k přístupu k textům náležejícím záznamům je potřeba příslušná aktivace.
- enabled – Tento atribut slouží k ověření toho, že daná instance objektu je pro program aktivní (může ji využít), nebo není (zachází se s ní jako, kdyby neexistovala).
- price – V tomto atributu se uchovává cena modulu.
- urlPrefixex – Kolekce urlPrefixes uchovává prefixy adres, které spadají pod daný modul.
- description – Atribut description uchovává HTML kód obsahující naformátovaný popis modulu, který se zobrazí v objednávkovém formuláři při zobrazení více informací o modulu.

Třída UrlPrefix mapuje prefixy adres, které se využívají k zjištění toho, jestli daný záznam spadá pod daný modul. Tato třída má atributy:

- id – Tento atribut je jedinečný identifikátor instance objektu.
- value – Zde je uchována hodnota prefixu adresy.

PreselectedModulesPackage je třída, která slouží k reprezentaci předvoleného balíčku modulů a má vazbu na moduly. Obsahuje atributy:

- id – Tento atribut je jedinečný identifikátor instance objektu.
- name – Zde je uložen název modulu.
- enabled – Tento atribut slouží k ověření toho, že daná instance objektu je pro program aktivní (může ji využít), nebo není (zachází se s ní jako, kdyby neexistovala).
- price – V tomto atributu se uchovává cena předvoleného balíčku.
- description – Atribut description uchovává HTML kód obsahující naformátovaný popis předvoleného balíčku, který se zobrazí v objednávkovém formuláři při zobrazení více informací o předvoleném balíčku.
- modules – Tato kolekce uchovává moduly, které daný předvolený balíček sdružuje.

3.2.2 Zabezpečení aplikace

Při zabezpečení aplikace je potřeba vytvořit spousty funkcionalit, které jsou pro většinu systémů stejné. Mnohé webové systémy obsahují funkcionality: přihlášení, odhlášení, resetování hesla, registraci, zobrazení profilu, editaci profilu a změna hesla. Pro Symfony existuje balíček, jehož implementací do systému se vyřeší všechny vypsané funkcionality. Zmínění balíček se jmenuje FosUserBundle a patří k jedněm z nejpoužívanějších Symfony balíčků vůbec. Po jeho stažení pomocí technologie Composer je potřeba ho aktivovat v souboru AppKernel.php (Ukázka PHP kódu 8). Tento balíček obsahuje řadiče, služby, šablony atd., které zajišťují chod zmíněných funkcionalit. Po jeho aktivaci v AppKernel.php a nastavení v konfiguračních souborech security.yml a config.yml je vše připraveno k použití. Balíček je plně konfigurovatelný a je možné velmi mnoho věcí změnit (např. šablony – ukázáno dále v této kapitole).

Ukázka PHP kódu 8 – Aktivace balíčku FosUserBundle v AppKernel.php

```php
<?php

use ...

class AppKernel extends Kernel
{
    public function registerBundles()
    {
        $bundles = [
            new Symfony\Bundle\FrameworkBundle\FrameworkBundle(),
            new Symfony\Bundle\SecurityBundle\SecurityBundle(),
            new Symfony\Bundle\TwigBundle\TwigBundle(),
            new Symfony\Bundle\MonologBundle\MonologBundle(),
            new Symfony\Bundle\SwiftmailerBundle\SwiftmailerBundle(),
            new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
            new Sensio\Bundle\FrameworkExtraBundle\SensioFrameworkExtraBundle(),
            new EpisBundle\EpisBundle(),
            new \FOS\UserBundle\FOSUserBundle(),
            code
        ];
```

(Zdroj: Autor)

Dále je potřeba nastavit soubor security.yml. Jeho nastavení pro projekt EPIS® Online je zobrazeno na obrázku 15 a v podstatě se jedná o defaultní nastavení balíčku, které je možné nalézt v jeho dokumentaci. To dokazuje, že tento balíček je použitelný pro reálné byznysové projekty i v jeho nejjednodušší podobě. Nejdříve je potřeba říci v oddíle encoders, jaký se použije algoritmus pro šifrování hesel. V EPIS® Online je využit

algoritmus bcrypt. Dále je nastavena hierarchie rolí, v níž jdou přednosti takto: ROLE_USER < ROLE_ADIM < ROLE_SUPER_ADMIN. Tato hierarchie rolí však v EPIS® Online není využita, jelikož veškerá administrace obsahu (textů právních předpisů, uživatelů, licencí atd.) je řešena v jiném systému. V oddíle providers se udává, že se využije komponenta z FosUserBundle balíčku (tato komponenta slouží k načítání uživatelů z databáze). V posledním oddíle firewalls se nastavuje samotné zabezpečení aplikace. Je zde nastaveno, jaké adresy se využívají k přihlašování a odhlašování uživatelů (fos_user_security_login a fos_user_security_logout jsou názvy těchto cest). Posledním zajímavým nastavením je zde konfigurace funkcionality, která si pamatuje přihlášení uživatele, pokud si tak přeje, třeba i celý rok.

Obrázek 15 – Nastavení balíčku FosUserBundle v souboru security.yml

```yaml
security:
    encoders:
        FOS\UserBundle\Model\UserInterface: bcrypt

    role_hierarchy:
        ROLE_ADMIN:       ROLE_USER
        ROLE_SUPER_ADMIN: ROLE_ADMIN

    providers:
        fos_userbundle:
            id: fos_user.user_provider.username

    firewalls:
        main:
            pattern: ^/
            form_login:
                login_path:           fos_user_security_login
                provider:             fos_userbundle
                csrf_token_generator: security.csrf.token_manager
            logout:
                path: fos_user_security_logout
                invalidate_session: false
            anonymous:    true
            remember_me:
                secret:   "%secret%"
                lifetime: 31556926 # 1 year in seconds
                path:     /

    access_control:
```

(Zdroj: Autor)

V dalším kroku je potřeba vytvořit entitu, která bude reprezentovat uživatele v systému, jak je zobrazeno v ukázce PHP kódu 9. Tato entita musí dědit z třídy FOS\UserBundle\Model\User a to je vše, co je potřeba nastavit. Díky tomu tato třída získá všechny důležité atributy, které funkcionality FosUserBundle potřebují

(např. atributy, které využívá balíček FosUserBundle při obnově hesla). V ukázce je zobrazeno, jak do této třídy přidat vlastní atributy ($sessionID, $note) a vztahy mezi třídami ($activations, $licenses) a nastavit je pomocí anotací pro Doctrine.

Ukázka PHP kódu 9 - Entita User mapujícího uživatele v systému

```php
<?php

namespace EpisBundle\Entity\Main;

use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\ORM\Mapping as ORM;
use EpisBundle\Entity\Register\GlobalRegisterItem;
use EpisBundle\Model\Register\ParagraphsGroup;
use FOS\UserBundle\Model\User as FosUser;
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;
use Symfony\Component\Validator\Constraints as Assert;

/**
 * @ORM\Table(name="novyepis.manager_for_online_user")
 * @ORM\Entity(repositoryClass="EpisBundle\Repository\Main\UserRepository")
 */
class User extends FosUser
{
    /**
     * @var string
     * @ORM\Column(name="session_id", type="string", length=255, nullable=true)
     */
    private $sessionID;

    /**
     * @var string
     * @ORM\Column(name="note", type="text", nullable=true)
     */
    private $note;

    /**
     * @var ArrayCollection|Activation[]
     * @ORM\OneToMany(targetEntity="EpisBundle\Entity\Main\Activation", mappedBy="user",
     *      cascade={"all"}, orphanRemoval=true, fetch="EXTRA_LAZY")
     */
    private $activations;

    /**
     * @var ArrayCollection|License[]
     * @ORM\ManyToMany(targetEntity="EpisBundle\Entity\Main\License", mappedBy="licenseManagers",
     *      cascade={"all"}, fetch="EXTRA_LAZY")
     */
    private $licenses;

    Getters and setters
}
```

(Zdroj: Autor)

Balíček FosUserBundle je potřeba nastavit v souboru config.yml (Obrázek 16). Zde se nastavuje, jakým způsobem jsou uživatelé uloženi. EPIS® Online využívá MySQL databázi a objektově-relační mapování (Doctrine). Dále se zde nastavuje jméno zabezpečení, které má využít (Obrázek 15) a název třídy reprezentující uživatele systému (Ukázka PHP kódu 9).

Obrázek 16 – Nastavení balíčku FosUserBundle v souboru config.yml

```
fos_user:
    db_driver: orm # other valid values are 'mongodb', 'couchdb' and 'propel'
    firewall_name: main
    user_class: EpisBundle\Entity\Main\User
```

(Zdroj: Autor)

FosUserBundle využívá vlastní adresy akcí, takže přihlášení (pokud toto nastavení není změněno) probíhá na adrese /login a odhlášení probíhá na adrese /logout. Tyto adresy je možné změnit pomocí souboru routing.yml, který se nachází ve stejné složce s dalšími konfiguračními soubory (config.yml, security.yml atd.). Příloha 6 ukazuje, jak tuto změnu provést a jednotlivé adresy počeštit. Jelikož šablony FosUserBundle nejsou nastylovány pomocí CSS, tudíž je možné je přepsat a tím zachovat jednotný styl aplikace. To je popsáno v kapitole (Popis obrazovek). Po provedení kroků popsaných výše je možné využívat vestavěných bezpečnostních metod (je možno kontrolovat v akci řadiče, jestli je uživatel přihlášený a má příslušné role atd.). EPIS® Online využívá konkrétně komponentu Voter k zabezpečení obsahu textů jednotlivých právních předpisů, která je popsána v kapitole dále (Zabezpečení obsahu textu pomocí komponenty Voter).

3.2.3 Zajištění unikátnosti přístupů

Jedním z funkčních požadavků na systém bylo zabezpečení proti možnosti přihlášení na jeden účet. Toho bylo dosažena tak, že při přihlášení uživatele do systému se ukládá identifikátor session, která probíhá mezi prohlížečem uživatele a serverem, do databáze (konkrétně do tabulky reprezentující samotného uživatele). Potom systém naslouchá na jakýkoliv požadavek od uživatele a pokud se neshoduje identifikátor současné session, s tím, který má uživatel u sebe uložený v databázi, je odhlášen. Nejlépe je to vidět na následujícím příkladu. Uživatel 1 se přihlásí k danému účtu. Do databáze k tomuto účtu je uložen identifikátor session, kterou má mezi sebou uživatel 1 a webový server. Nyní systém naslouchá a při každém požadavku uživatele 1 kontroluje, jestli se identifikátor session shoduje s tím v databázi. Následně se přihlásí na stejný účet uživatel 2. Do databáze je zapsána hodnota identifikátoru jeho session. Uživatel 1 je při příštím požadavku na webový server odhlášen, protože ten zjistí, že se identifikátor session mezi uživatelem 1 a webovým serverem neshoduje s hodnotou uloženou v databázi.

V ukázce PHP kódu 10 je zobrazena třída LoginSubscriber, která implementuje rozhraní EventSubscriberInterface (toto rozhraní je potřeba implementovat, pokud je požadováno, aby třída uměla naslouchat na události během požadavku od uživatele). Té je pomocí konstruktoru předána služba umožňující aktualizaci entity User. Metoda getSubscribedEvents vrací asociativní pole hodnot, v kterém klíče reprezentují názvy událostí (přihlášení pomocí formuláře a automatické přihlášení při registraci), kterým třída naslouchá a hodnoty jsou názvy metod třídy, která se mají při události zavolat. Při zmíněných událostech je zavolána metoda onLogin, která dostává v parametru instanci objektu, který obsahuje údaje o dané události. Nejdříve se z této instance objektu pomocí metody dostane instance uživatele, který se přihlásil a objekt reprezentující současnou session. Následně je uživateli pomocí metody uložena hodnota identifikátoru session do atributu a tato hodnota je zapsána do databáze.

Ukázka PHP kódu 10 – Třída LoginSubscriber

```php
namespace EpisBundle\EventListener;

use EpisBundle\Entity\Main\User;
use FOS\UserBundle\Event\UserEvent;
use FOS\UserBundle\FOSUserEvents;
use FOS\UserBundle\Model\UserManagerInterface;
use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use Symfony\Component\Security\Http\SecurityEvents;

class LoginSubscriber implements EventSubscriberInterface
{
    /** @var UserManagerInterface */
    private $userManager;

    public function __construct(UserManagerInterface $userManager) { $this->userManager = $userManager; }

    public static function getSubscribedEvents()
    {
        return [
            FOSUserEvents::SECURITY_IMPLICIT_LOGIN      => "onLogin",
            SecurityEvents::INTERACTIVE_LOGIN           => "onLogin",
        ];
    }

    public function onLogin($event)
    {
        /** @var User $user */
        $user = $event instanceof UserEvent ? $event->getUser() : $event->getAuthenticationToken()->getUser();
        $session = $event->getRequest()->getSession();
        $user->setSessionID($session->getId());
        $this->userManager->updateUser($user);
    }
}
```

(Zdroj: Autor)

Další třídou, kterou je nutno pro dosažení funkcionality popsané výše vytvořit, je KernelRequestSubscriber (Příloha 7), která opět implementuje rozhraní EventSubscriberInterface. Konstruktorem třídy se předávají služby systému, které jsou důležité v metodách třídy. Instance této třídy naslouchá na každý požadavek na webovou aplikaci (Ukázka PHP kódu 11).

Ukázka PHP kódu 11 – Metoda getSubscribedEvents třídy KernelRequestSubscriber

```php
public static function getSubscribedEvents()
{
    return [
        KernelEvents::REQUEST => "onKernelRequest"
    ];
}
```

(Zdroj: Autor)

Metoda onKernelRequest (Ukázka PHP kódu 12) opět dostává instanci objektu, která obsahuje užitečné informace o události. Nejdříve se v této metodě zjišťuje, jestli je přihlášený uživatel. Pokud není, metoda končí. Jinak se pokračuje dál a pomocí služeb, které byly této instanci předány konstruktorem, se získává instance přihlášeného uživatele (entita User) a identifikátor současné session. Pokud se identifikátor současné session shoduje s hodnotou, která je uložená v databázi, metoda končí. Pokud tomu tak ale není, dochází k odhlášení uživatele (je přesměrován na adresu odhlášení) a je mu vypsána zpráva, která ho informuje o nastalé situaci.

Ukázka PHP kódu 12 – Metoda onKernelRequest třídy KernelRequestSubscriber

```php
public function onKernelRequest(GetResponseEvent $event)
{
    if (!$this->isUserLoggedIn())
    {
        return;
    }
    /** @var User $user */
    $user = $this->tokenStorage->getToken()->getUser();
    $sessionID = $this->session->getId();
    if ($sessionID === $user->getSessionID())
    {
        return;
    }
    $this->session->getFlashBag()->add('danger', 'Byl jste odhlášen, protože došlo k přihlášení na tento účet z jiného zařízení.');
    $response = new RedirectResponse($this->urlGenerator->generate( name: "fos_user_security_logout"));
    $event->setResponse($response);
}
```

(Zdroj: Autor)

Metoda isUserLoggedIn (Ukázka PHP kódu 13) je volána v metodě popsané výše. Tato metoda kontroluje, jestli je do systému přihlášený nějaký uživatel. Využívá k tomu službu předanou konstruktorem třídy (konkrétně její metodu isGranted, kdy se dotazuje, jestli je uživatel přihlášený).

Ukázka PHP kódu 13 - Metoda isUserLoggedIn třídy KernelRequestSubscriber

```php
private function isUserLoggedIn()
{
    try
    {
        return $this->authorizationChecker->isGranted( attributes: 'IS_AUTHENTICATED_REMEMBERED');
    }
    catch (AuthenticationCredentialsNotFoundException $exception)
    {
        // Ignoring this exception.
    }

    return false;
}
```

(Zdroj: Autor)

Aby obě dvě třídy zmíněné výše byly zaregistrovány a mohly naslouchat na popsané události, je nutné je správně nakonfigurovat v souboru services.yml (Obrázek 17). Konkrétně je oběma třídám přidám příslušný tag, který určuje, že implementují rozhraní EventSubscriberInterface, a mohou naslouchat událostem.

Obrázek 17 – Konfigurace tříd implementující EventSubscriberInterface v services.yml

```yaml
EpisBundle\EventListener\LoginSubscriber:
    tags:
        - { name: kernel.event_subscriber }

EpisBundle\EventListener\KernelRequestSubscriber:
    tags:
        - { name: kernel.event_subscriber }
```

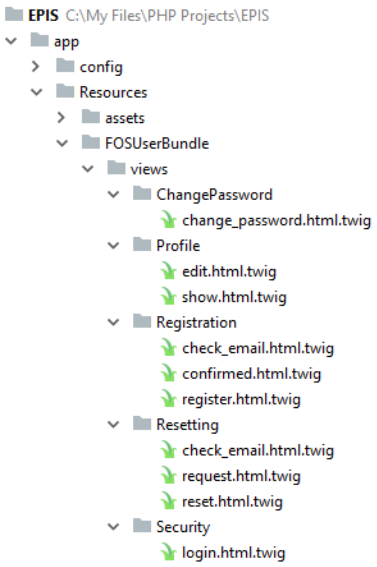
(Zdroj: Autor)

3.2.4 Popis obrazovek

Všechny obrazovky systému popsané v této práci rozšiřují základní šablonu (Příloha 8). Ta obsahuje hlavičku (menu aplikace, fulltextové vyhledávání atd.) a patičku (odkazy na sociální sítě, obchodní podmínky atd.). Mezi tyto dva elementy jsou vkládány obsahy jednotlivých šablon systému.

Jelikož jsou funkcionality zabezpečení řešeny pomocí balíčku FosUserBundle, je nutné pro zachování jednotného grafického rozložení aplikace přepsat šablony tohoto balíčku. To se provede velmi jednoduše. Stačí ve složce app\Resources vytvořit složku FosUserBundle a vytvořit v ní strukturu, která je zobrazena na obrázku 18. Tím se zajistí to, že FosUserBundle bude využívat šablony z tohoto úložiště, a ne své defaultní.

Obrázek 18 – Přepsaní defaultních šablon balíčku FosUserBundle



(Zdroj: Autor)

Příloha 9 zobrazuje Twig šablonu přihlašovacího formuláře. Tato šablona po vykreslení (Příloha 19) webovým prohlížečem obsahuje dva formulářové vstupy – email a heslo. Pod těmito vstupy se nachází možnost volby, jestli si má aplikace pamatovat přihlášení. Posledním prvkem této obrazovky je odkaz na funkcionalitu umožňující změnu zapomenutého hesla.

Další obrazovkou aplikace je registrační formulář (šablona - Příloha 10, obrazovka - Příloha 20). Při registraci do systému je nutné zadat email, který zároveň slouží jako přihlašovací jméno do systému a heslo, které je nutné potvrdit jeho opětovným zapsáním. Po úspěšném vyplnění registračního formuláře je uživateli vytvořen účet a je

přesměrován na obrazovku oznamující tuto skutečnost (šablona - Příloha 11, obrazovka - Příloha 21). Zde je uživateli oznámeno, že mu byla na zadanou emailovou adresu odeslána zpráva obsahující odkaz, na který musí kliknout, aby dokončil registraci. Kliknutím na tento odkaz je přesměrován na obrazovku, která mu oznamuje, že jeho účet byl aktivován (šablona - Příloha 12, obrazovka - Příloha 22).

Uživatel má možnost zobrazit si svůj profil v systému (šablona - Příloha 13, obrazovka - Příloha 23). Jelikož firma využívá k fakturaci jiný systém, tak se v EPIS® Online ukládá o uživatelích minimum informací. Zobrazený profil je možné editovat příslušným formulářem (šablona - Příloha 14, obrazovka - Příloha 24). Uživatel také může změnit své heslo (šablona Příloha 15, obrazovka - Příloha 25).

Při zapomenutí hesla je uživateli dána možnost, aby si toto heslo změnil. Nejdříve musí vyplnit žádost o změnu zapomenutého hesla (šablona - Příloha 16, obrazovka - Příloha 26), kde vyplní svůj email. Po vyplnění žádosti je přesměrován na oznámení, že jeho žádost byla vyřízena (šablona - Příloha 17, obrazovka - Příloha 27). Zde mu je oznámeno, že má navštívit email, který zadal při registraci, jelikož mu tam byla odeslána zpráva s odkazem, který mu umožní nastavení nového hesla. Po kliknutí na tento odkaz je přenesen na formulář umožňující zadání hesla (šablona - Příloha 18, obrazovka - Příloha 28)

3.3 Modul právních předpisů ČR

Právní předpisy ČR jsou pro firmu nejdůležitější modul, jelikož jeho prodej tvoří podstatnou část zisků firmy a patří k nejoblíbenějším mezi zákazníky. Firma zpracovává texty právních předpisů, které vycházejí v tištěných sbírkách do elektronické podoby, čím přináší koncovému uživateli různé výhody:

- Vyhledávání podle různých atributů a vztahů
- Možnost přidávání předpisů a jejich jednotlivých odstavců do oblíbených položek, čím se zajistí zjednodušený přístup
- Možnost psaní poznámek k předpisům a jejich jednotlivým odstavcům
- Porovnávání verzí jednotlivých odstavců (nové znění odstavce, které přinesla novela a staré původní znění odstavce) (Příloha 29)
- Provázanost textů předpisů (když se v textu předpisu píše o nějakém jiném předpisu, tak se zde nachází hypertextový odkaz, který uživatele nasměruje na tento předpis) (Obrázek 19)

Obrázek 19 – Ukázka provázanosti textů

§6

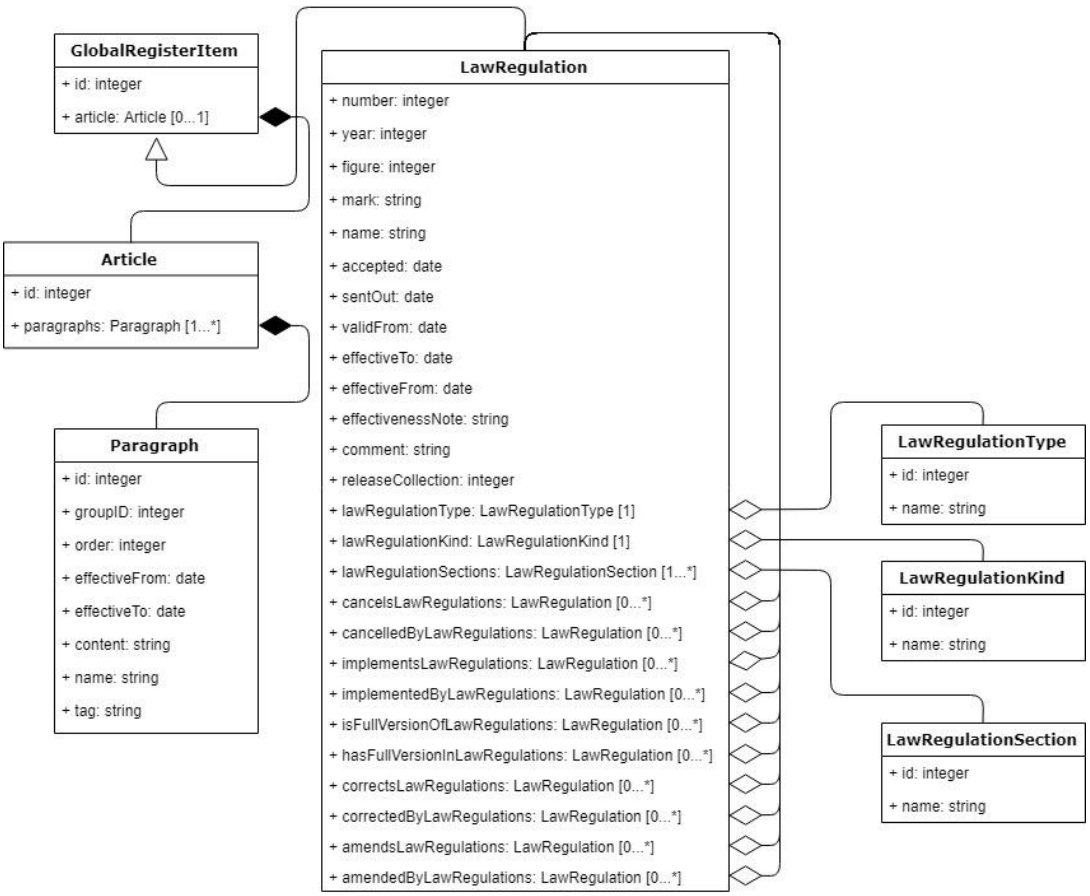
§6 zrušen právním předpisem č. 313/2002 Sb.

(Zdroj: https://novyepis.cz)

3.3.1 Datový model

Na diagramu níže (Obrázek 20) je zobrazeno schéma tříd modulu právních předpisů ČR. Toto schéma je následně vytvořeno pomocí entit a technologie Doctrine, čímž se vytvoří potřebné schéma tabulek v databázi.

Obrázek 20 - Diagram tříd modulu právních předpisů ČR



(Zdroj: Autor)

GlobalRegisterItem je třída, která je rodičem všech záznamů, které se v EPIS® Online evidují. Z této třídy dědí třída reprezentující finanční zpravodaj MF ČR, judikát, vyhlášku města atd. Také z ní dědí třída LawRegulation a dále může mít vazbu na třídu

Article. To znamená, že se v systému mohou evidovat např. právní předpisy ČR, které nemají zadaný jejich obsah (samotné znění předpisu), ale v systému jsou evidovány základní informace o nich (od kdy platí, co je ruší atd.).

Třída Article reprezentuje záznam s textem. Nezáleží na tom, co tento text obsahuje a k čemu se vztahuje. Tato třída slouží ke sdružení jednotlivých odstavců, z kterých se sestavují texty jednotlivých záznamů v systému.

Třída Paragraph znázorňuje odstavec textu, z kterých se dohromady utvoří znění nějakého záznamu. Obsahuje atributy:

- id – Tento atribut je jedinečný identifikátor instance objektu.
- groupID – Tento atribut slouží k sdružení jednotlivých verzí odstavců. Pokud se do systému zadá nové znění nějakého odstavce, tak dostane hodnotu do tohoto atributu stejnou jako mělo předešlé znění odstavce. V rámci skupiny je pak možné najít aktuální, minulá a budoucí znění odstavce (v závislosti na datu, ke kterému se dotaz vztahuje).
- order – Atribut order značí pořadí odstavce na sestavené stránce. Sdružené odstavce pomocí groupID mají hodnotu tohoto atributu stejnou.
- effectiveFrom – Hodnota tohoto atributu obsahuje datum, které znázorňuje, odkdy odstavec platí. Může nabývat hodnotu NULL.
- effectiveTo – V tomto atributu je uloženo datum znázorňující, dokdy odstavec platí. Může nabývat hodnotu NULL.
- Content – Tento atribut uchovává obsah odstavce.
- Name – Atribut name obsahuje název odstavce.
- Tag – Textový atribut, který je využit pro interní potřeby firmy.

Právní předpis ČR je v systému modelován třídou LawRegulation. Tato třída dědí z třídy GlobalRegisterItem a obsahuje velké množství atributů:

- id – Tento atribut je jedinečný identifikátor instance objektu.
- number – Tento atribut uchovává číslo předpisu.
- year – Zde je uložen rok předpisu.
- figure – Atribut figure uchovává částku.
- mark – Textový atribut uchovávající označení předpisu.
- name – V tomto atributu se eviduje název předpisu.
- accepted – Hodnota tohoto atributu znázorňuje datum přijetí předpisu.
- sentOut – Atribut sentOut eviduje datum, kdy byl předpisu rozeslán.

- validFrom – Tento atribut ukládá datum, odkdy předpis platí.
- effectiveTo – Zde se uchovává datum, dokdy je předpis účinný. Může být NULL.
- effectiveFrom – Atribut eviduje datum, odkdy je předpis účinný. Může nabývat hodnoty NULL.
- effectivenessNote – Textový atribut, který uchovává poznámku k účinnosti.
- comment – Do tohoto atributu se ukládá hodnota majáku (Dodatečná redakční informace k předpisu).
- releaseCollection – Tento atribut slouží k rozlišení původu předpisu (sbírka zákonů ČR, sbírka mezinárodních smluv atd.)
- lawRegulationType a lawRegulationKind – Zde se eviduje druh předpisu, akorát první atribut je obecně a druhý více podrobně.
- lawRegulationSections – Tato kolekce uchovává oblasti úpravy.
- cancelsLawRegulations – Kolekce obsahující předpisy, který daný předpis ruší.
- canelledByLawRegulations – Kolekce předpisů, které daný předpis ruší.
- implementsLawRegulations – Kolekce předpisů, které daný předpis provádí.
- implementedByLawRegulations – Kolekce předpisů, kterými je daný předpis prováděn.
- isFullVersionOfLawRegulations – Kolekce předpisů, které jsou plným zněním daného předpisu.
- hasFullVersionOfLawRegulations – Kolekce předpisů, v kterých má daný předpis plné znění.
- correctsLawRegulations – Kolekce předpisů, které daný předpis opravuje.
- correctedByLawRegulations – Kolekce předpisů, které opravují daný předpis.
- amendsLawRegulations – Kolekce předpisů, které daný předpis novelizuje.
- amendedByLawRegulations – Kolekce předpisů, které novelizuje daný předpis.

Třídy LawRegulationType (znázorňuje druh předpisu obecně), LawRegulationKind (znázorňuje druh předpisu podrobně) a LawRegulationSection (znázorňuje oblast úpravy) jsou jednoduchými třídami, které obsahují identifikátor instance objektu a atribut, který uchovává jejich textový název.

3.3.2 Vyhledávání v předpisech

Vyhledávání v právních předpis je uživatelům přístupné pomocí vyhledávacího formuláře, který vyhledává pomocí některých výše popsaných atributů a vazeb (Příloha

30). Vytvořit takovýto formulář, z kterého se po jeho odeslání vygeneruje SQL dotaz pro vyhledání kýžených právních předpisů ČR, může být pro vývojáře velmi časově náročné. Z tohoto důvodu byl v EPIS® Online využit balíček LexikFormFilterBundle, který tuto proceduru velmi urychluje. Tento balíček obsahuje již předpřipravené formulářové komponenty, které umí správně vyhledávací SQL dotaz sestavit.

V ukázce PHP kódu 14 je zobrazena definice vyhledávacího formuláře, který si konstruktorem žádá službu umožňující spravovat entity. Dále je zde vidět defaultní nastavení formuláře (metoda, datová třída, validační skupina a vypnutí ochrany proti CSRF útokům).

Ukázka PHP kódu 14 – Definice třídy LawRegulationFilterType

```php
namespace EpisBundle\Form\Register\Filter;

use ...

class LawRegulationFilterType extends AbstractType
{
    /** @var EntityManagerInterface */
    protected $entityManager;
    /**
     * LawRegulationFilterType constructor.
     * @param EntityManagerInterface $entityManager
     */
    public function __construct(EntityManagerInterface $entityManager)
    {
        $this->entityManager = $entityManager;
    }

    public function buildForm(FormBuilderInterface $builder, array $options){...}

    public function getBlockPrefix(){...}

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults(array(
            "csrf_protection"   => false,
            "method"            => "GET",
            "data_class"        => LawRegulation::class,
            "validation_groups" => ["filtering"]
        ));
    }
}
```

(Zdroj: Autor)

Metoda buildForm třídy LawRegulationFilterType se stará o vytvoření vyhledávacích elementů formuláře (Příloha 31). Do formuláře přidává políčka, která se vztahují na atributy třídy LawRegulation a nastavuje k nim příslušné vyhledávací prvky

z balíčku LexikFormFilterBundle. Například přidává políčko pro vyhledávaní v atributu number, které uchovává číselný datový typ a příslušnému políčku nastavuje vyhledávací prvek NumberFilterType, který umožňuje vyhledávání v číselných hodnotách a následně umí správně upravit SQL dotaz na databázi. Jak je vidět ve zmíněné příloze, balíček LexikFormFilterBundle obsahuje prvky pro vyhledávání v číselných, textových, datumových a dalších datových typech a umí vyhledávat ve vazbách mezi entitami, pomocí EntityFilterType, což je také zobrazeno v příloze.

Vyhledávání v systému postupuje následujícím způsobem. Samotný vyhledávací formulář má separátní akci řadiče na adrese https://novyepis.cz/pravni-predpisy/vyhledavani. Tato akce je zobrazena v ukázce PHP kódu 15. Nejdříve se vytvoří instance vyhledávacího formuláře, který si načte data z požadavku uživatele. Když není formulář odeslán, nebo validní, je vrácena vykreslená šablona, která tento formulář uživateli zobrazí. Pokud však je formulář odeslán i validní, je uživatel přesměrován na akci, která zobrazí stránkované výsledky vyhledávaní.

Ukázka PHP kódu 15 – Akce řadiče zobrazující vyhledávací formulář

```php
/**
 * @Route("/pravni-predpisy/vyhledavani", name="epis_law_regulation_search")
 * @Breadcrumb("Vyhledávání v právních předpisech ČR")
 * @Method("GET")
 */
public function searchAction(Request $request)
{
    $searchForm = $this->createForm( type: LawRegulationFilterType::class,
        data: null,
        ["validation_groups" => ["filtering"]]);
    $searchForm->handleRequest($request);
    if ($searchForm->isSubmitted() && $searchForm->isValid())
    {
        return $this->redirectToRoute( route: "/pravni-predpisy/vysledky-vyhledavani", $request->query->all());
    }

    return $this->render( view: "@Epis/Register/LawRegulation/search", array(
        "search_form" => $searchForm->createView()
    ));
}
```

(Zdroj: Autor)

Obrazovka se stránkovanými výsledky vyhledávání obsahuje i vyhledávací formulář, který uživatel může využít k upravení vyhledávacího dotazu (Příloha 32). V ukázce PHP kódu 16 je zobrazena akce řadiče, která se stará o zobrazení vyhledaných výsledků. Opět se tvoří instance vyhledávacího formuláře, která si načte data z požadavku uživatele. Dále se do proměnné repository ukládá repositář entity LawRegulation a do proměnné queryBuilder se tímto repositářem tvoří instance třídy, která pomáhá sestavovat vyhledávací SQL dotaz. Následuje podmínka, která kontroluje,

jestli byl formulář odeslán a je validní. Pokud tomu tak je, se volá služba patřící k balíčku LexikFormFilterBundle, která dle dat z formuláře nastaví instanci třídy v proměnné queryBuilder příslušná vyhledávací omezení (upraví se SQL dotaz). Následuje další podmínka, která využívá instanci služby Paginator (tato služba ulehčuje stránkování vyhledaných výsledků). V podmínce se dotazuje, jestli vyhledané výsledky vyhledávání jsou nějak seřazené (instance služby Paginator má přístup k instanci třídy Request, takže tuto informaci dokáže zjistit). Pokud uživatel nezadal žádné řazení, repositář upraví vyhledávací SQL dotaz přidáním defaultního řazení. Následně služba Paginator vyhledávací SQL dotaz také upraví a přidá k němu stránkování (pomocí SQL syntaxe LIMIT a OFFSET). Do proměnné response se následně ukládá vytvořená šablona, která zobrazuje výsledky vyhledávání s vyhledávacím formulářem a je jí do hlavičky nastaven cookie soubor, díky kterému si aplikace pamatuje, kolik výsledků vyhledávání chtěl uživatel naposledy zobrazit.

Ukázka PHP kódu 16 – Akce řadiče zobrazující výsledky vyhledávání

```php
/**
 * @Route("/pravni-predpisy/vysledky-vyhledavani", name="epis_law_regulation_search_results")
 * @Breadcrumb("Vyhledávání v právních předpisech ČR", routeName="epis_law_regulation_search")
 * @Breadcrumb("Výsledky vyhledávání v právních předpisech ČR")
 * @Method("GET")
 */
public function searchResultsAction(Request $request, Paginator $paginator)
{
    $searchForm = $this->createForm( type: LawRegulationFilterType::class,
        data: null,
        ["validation_groups" => ["filtering"]]);
    $searchForm->handleRequest($request);
    $repository = $this->getDoctrine()->getRepository( persistentObject: LawRegulation::class);
    $queryBuilder = $repository->createQueryBuilder( alias: "law_regulation");
    if ($searchForm->isSubmitted() && $searchForm->isValid())
    {
        $this->get("lexik_form_filter.query_builder_updater")->addFilterConditions($searchForm, $queryBuilder);
    }
    if (!$paginator->isSorted())
    {
        $repository->addOrderingRules($queryBuilder, lawRegulationAlias: "law_regulation");
    }
    /** @var LawRegulation[]|PaginationInterface $pagination */
    $pagination = $paginator->createPagination($queryBuilder);
    $response = $this->render( view: "@Epis/Register/LawRegulation/search_results", array(
        "law_regulations_pagination" => $pagination,
        "search_form"                => $searchForm->createView()
    ));
    $response->headers->setCookie($paginator->createLimitCookie());

    return $response;
}
```

(Zdroj: Autor)

3.3.3 Zobrazení konkrétního předpisu

Datový model umožňuje evidovat verze jednotlivých odstavců zákonů. Ty se mohou měnit jinými zákony, či novelami. EPIS® Online z tohoto důvodu dovoluje uživateli sestavit právní předpis ČR ke zvolenému datu, což je zobrazeno v příloze 33. Zde uživatel zadá datum a následně je přesměrován na akci řadiče zobrazující právní předpis (Příloha 34). Datum sestavení předpisu je předán akci pomocí metody GET. V této akci se nejdříve kontroluje, jestli má právní předpis vazbu na třídu Article. Pokud jí nemá, je uživateli oznámeno, že zadaný předpis nemá v systému plné znění. Dále se z požadavku uživatele získává datum sestavení. Pokud není žádné datum předané, využije se dnešní datum. Tato hodnota se následně musí zkontrolovat, protože datum sestavení musí být větší než datum, od kdy je předpis účinný a zároveň musí být menší než hodnota, do kdy je právní předpis účinný (pokud má předpis tuto hodnotu zadanou). Následuje vykreslení šablony, která zobrazuje právní předpis (Příloha 35). Šabloně je předán právní předpis, datum sestavení, odstavce právního předpisu a hodnota, která udává, jestli se má text skrýt. Skrytý text je k zobrazení v příloze 5. Zjištění této hodnoty je popsáno v kapitole Zabezpečení obsahu textu pomocí komponenty Voter. Šabloně se předávají separátně odstavce kvůli výkonu aplikace. Některé předpisy mají mnoho novel (tudíž i mnoho verzí odstavců), a proto se speciálním SQL dotazem vyhledávají jen ty, které jsou označené za platné k předanému datu sestavení předpisu. Metoda vyhledávající tyto odstavce je zobrazena v ukázce PHP kódu 17. Tato metoda si bere za argumenty právní předpis a datum sestavení a na základě těchto hodnot sestaví specifický SQL dotaz, který Doctrine neumí sestavit. Následně jsou výsledky tohoto SQL dotazu namapovány na instance třídy Paragraph.

Ukázka PHP kódu 17 – Metoda getLawRegulationParagraphs

```php
private function getLawRegulationParagraphs($lawRegulation, $compilationDate)
{
    $sql = "SELECT id, gid, ord, tags, comment, content, rootid, hidden, deleted,
            (CASE WHEN fromdate IS NULL THEN ? ELSE fromdate END) AS fromdate,
            (CASE WHEN todate   IS NULL THEN ? ELSE todate   END) AS todate
            FROM paragraphs
            WHERE rootid=? AND rootid IS NOT NULL AND ((fromdate IS NULL OR fromdate <= ?) AND (todate IS NULL OR todate >= ?))
            ORDER BY ord ASC";
    $mapping = $this->getResultSetMapping();
    $query = $this->getEntityManager()->createNativeQuery($sql, $mapping);
    $query->setParameter( key: 1, $lawRegulation->getEffectiveFrom(), type: Type::DATE);
    $query->setParameter( key: 2, $lawRegulation->getEffectiveTo(), type: Type::DATE);
    $query->setParameter( key: 3, $lawRegulation->getArticle()->getRootID());
    $result = $query->getResult();

    return $result;
}
```

(Zdroj: Autor)

3.3.4 Zabezpečení obsahu textu pomocí komponenty Voter

Zabezpečení textů jednotlivých záznamů v systému probíhá pomocí skrytí jednotlivých znaků (Příloha 5). Jestli k tomuto skrytí má dojít rozhoduje komponenta Voter. K vytvoření vlastní komponenty Voter je potřeba vytvořit třídu, která dědí z třídy Voter a následně je nutné přepsat její metody supports a voteOnAttribute vlastní logikou (Příloha 36). Voter je volán pomocí metode isGranted, která je dostupná v akcích řadičů (Příloha 34). Metodě se předává textová hodnota reprezentující probíhající akci (nejčastěji se používají „VIEW“, „EDIT“ a „DELETE“) a instanci objektu, na které se má akce provést. Při volání této metody se zavolají všechny komponenty Voter, které jsou v systému vytvořeny a pomocí metody supports, se rozhodne, jestli daný Voter bude rozhodovat o požadované akci nad danou instancí objektu, nebo ne. Pokud je rozhodnuto, že ano, tak se zavolá metoda voteOnAttribute.

Jak bylo napsáno výše, metoda supports je volána ve chvíli, kdy se určuje, jestli se k rozhodnutí má využít daná instance komponenty Voter. Metodou isGranted jsou jí předány dva argumenty – textová hodnota reprezentující danou akci a instance objektu, na které se má akce provést. V ukázce PHP kódu 18 se v první podmínce určuje, jestli předaná textová hodnota je danou komponentou Voter podporována. Pokud tomu tak není, vrací metoda false. V další podmínce probíhá kontrola datového typu argumentu subject. Pokud se nejedná o instanci třídy GlobalRegisterItem, metoda vrací false, jinak vrací true.

Ukázka PHP kódu 18 – Metoda supports třídy GlobalRegisterVoter

```php
protected function supports($attribute, $subject)
{
    // if the attribute isn't one we support, return false
    if (!in_array(mb_strtolower($attribute), array(self::VIEW)))
    {
        return false;
    }
    // only vote on Post objects inside this voter
    if (!$subject instanceof GlobalRegisterItem)
    {
        return false;
    }

    return true;
}
```

(Zdroj: Autor)

Moteda voteOnAttribute je volána ve chvíli, kdy metoda supports vrátí true (bude rozhodovat o prováděné akci nad instancí objektu). V ukázce PHP kódu 19 se nedříve argument reprezentující prováděnou akcí převede na hodnotu zapsanou malými písmeny a z proměnné token se pomocí její metody getUser vrátí instance objektu User, který je zrovna přihlášen. Pokud uživatel není přihlášen, tato metoda vrací textovou hodnotu „Anon.“. Následuje kontrola argumentu attribute, jestli se shoduje s konstantou třídy reprezentující akci pro zobrazení. Za touto podmínkou se nachází vyvolání výjimky LogicException, jelikož do této části kódu by se aplikace nikdy neměla dostat. V popsané podmínce se nachází další podmínka, která kontroluje, jestli nepatří záznam do modulu, který je uživatelům přístupný zdarma. Pokud tomu tak je, tak metoda vrací true, jinak se pokračuje dále v exekuci kódu na další podmínku, která kontroluje, jestli proměnná user neobsahuje hodnotu datového typu User (např. obsahuje textovou hodnotu „Anon.“). V takovém případě je návratová hodnota false, jelikož se jedná o záznam, který není v modulu, který je poskytován zdarma a uživatel není přihlášený, tudíž nemá právo si zobrazit text záznamu. Pokud je uživatel přihlášen, kód se nedostane do exekuce těla této podmínky a vrací se hodnota, kterou vrátí metoda isGlobalRegisterItemBought.

Ukázka PHP kódu 19 – Metoda voteOnAttribute třídy GlobalRegisterVoter

```php
protected function voteOnAttribute($attribute, $subject, TokenInterface $token)
{
    $attribute = mb_strtolower($attribute);
    /** @var GlobalRegisterItem $post */
    $globalRegisterItem = $subject;
    $user = $token->getUser();
    if ($attribute == self::VIEW)
    {
        if ($this->isGlobalRegisterItemForFree($globalRegisterItem))
        {
            return true;
        }
        if (!$user instanceof User)
        {
            return false;
        }

        return $this->isGlobalRegisterItemAccessBought($globalRegisterItem, $user);
    }

    throw new \LogicException('This code should not be reached!');
}
```

(Zdroj: Autor)

Metoda isGlobalRegisterItemForFree (Ukázka PHP kódu 20) rozhoduje o tom, jestli daný záznam patří do modulu, který uživatelé mají k dispozici zdarma, nebo tomu tak není. Pomocí konstruktorem předané služby mainObjectManager je získán repositář třídy Module, kterým je vytvořena instance třídy QueryBuilder. Tato instance třídy je využita k sestavení SQL dotazu, který hypotézu popsanou výše ověří.

Ukázka PHP kódu 20 – Metoda isGlobalRegisterItemForFree třídy GlobalRegisterVoter

```php
/**
 * @param GlobalRegisterItem $globalRegisterItem
 * @return bool
 */
private function isGlobalRegisterItemForFree($globalRegisterItem)
{
    $result = $this->mainObjectManager->getRepository( className: Module::class)
        ->createQueryBuilder( alias: "module")
        ->join( join: "module.urlPrefixes", alias: "url_prefixes")
        ->where( predicates: "module.enabled = :enabled")->setParameter( key: "enabled", value: true)
        ->andWhere("module.freeOfCharge = :free_of_charge")->setParameter( key: "free_of_charge", value: true)
        ->andWhere("url_prefixes.value = :url_prefix")->setParameter( key: "url_prefix", $this->getUrlPrefix())
        ->getQuery()->getResult();

    return !empty($result);
}
```

(Zdroj: Autor)

Metoda isGlobalRegisterAccessBought (Příloha 37) si bere za argument záznam, o kterém se rozhoduje a instanci uživatele, který je přihlášený. Opět je pomocí služby mainObjectManager získán repositář třídy User, kterým je vytvořena instance třídy QueryBuilder. Touto instancí je následně vytvořen SQL dotaz, který se propojí do uživateli přidělených aktivací licence. Z těchto aktivací následuje propojení do licencí, z kterých se dále propojuje do zakoupených modulů. U modulů se kontroluje, jestli současný prefix URL adresy je přiřazen k některému ze zakoupených modulů. Při propojení dotazem do licencí se také kontroluje, jestli jsou licence aktivovány a jestli jsou platné (porovnává se se datum platnosti od a datum platnosti do se současným datem). Metoda vrací true, pokud má uživatel k dispozici licenci splňující výše popsaná omezení, jinak metoda vrací false.

Pomocná metoda getUrlPrefix třídy GlobalRegisterVoter (Ukázka PHP kódu 21) je využita ve výše popsaných metodách (isGlobalRegisterAccessBought a isGlobalRegisterItemForFree). Tato metoda využívá konstruktorem načtené služby uchovávají informace o požadavku uživatele na server. Konkrétně z ní získává URL adresu, na kterou se uživatel dotazuje. Například když se uživatel dotazuje

na https://novyepis.cz/pravni-predpisy/77951, tato metoda z uvedené adresy vyseparuje hodnotu „/pravni-predpisy“, která je využita k dohledávaní modulů, ke kterým je tato hodnota vztažena.

Ukázka PHP kódu 21 – Metoda getUrlPrefix třídy GlobalRegisterVoter

```php
/**
 * @return string
 */
private function getUrlPrefix()
{
    $urlPrefix = parse_url($this->request->getUri(), component: PHP_URL_PATH);
    $urlPrefix = StringHelper::isNullOrWhiteSpace($urlPrefix) ? "" : $urlPrefix;
    $urlPrefix = StringHelper::substring($urlPrefix, startingIndex: 1);
    $index = StringHelper::indexOf($urlPrefix, needle: "/");
    if ($index > -1)
    {
        $urlPrefix = StringHelper::remove($urlPrefix, $index);
    }
    $urlPrefix = "/" . $urlPrefix;

    return $urlPrefix;
}
```

(Zdroj: Autor)

3.4 Modul platební brány

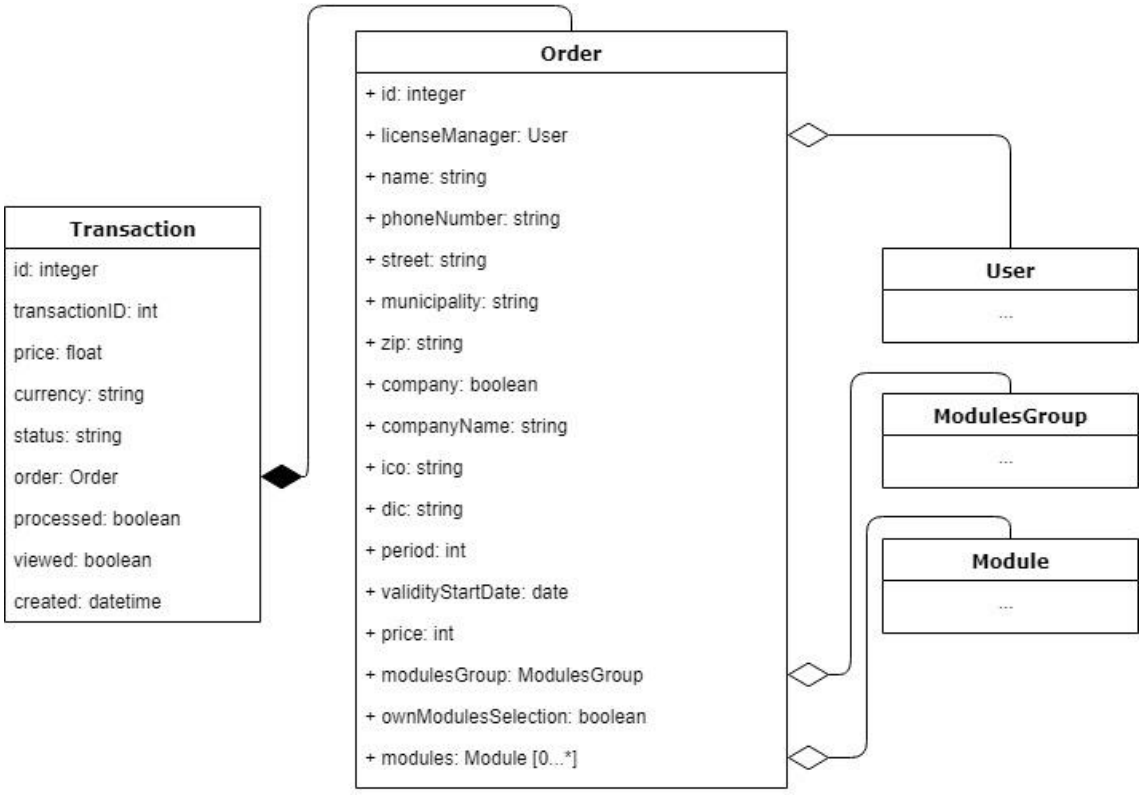
Modul platební brány slouží k propojení platební brány GoPay a EPIS® Online. Díky tomuto propojení si uživatel může zakoupit licenci k přístupu do systému přes objednávkový formulář, který je dostupný online. Tento objednávkový formulář má čtyři kroky. V prvním kroku si uživatel vybírá, jestli si zakoupí předvolený balíček modulů (Příloha 38), nebo si moduly navolí sám (Příloha 39). Může si zde také zobrazit detailní podpis jednotlivých nabízených předmětů. V následujícím druhém kroku (Příloha 40) probíhá volba období, na které si uživatel zakupuje přístup k předmětům, které si v předchozím kroku vybral. Nacházejí se zde čtyři předvolené možné doby předplatného (1 den, 30 dní, 180 dní a 365 dní). Uživatel má v tomto kroku také k dispozici vstup, kde může kromě těchto čtyř předvolených období vybrat vlastní. Zadaná hodnota musí být od 1 dnu do 365 dní. Při každé změně období dochází ke spojení se serverem a uživateli je přepočítána a zobrazena cena. V pravé části tohoto kroku se nachází rekapitulace. Po druhém kroku následuje třetí krok, kde se zadávají fakturační a doplňující údaje (Příloha 41 a Příloha 42). Opět se v pravé části formuláře nachází rekapitulace předchozích kroků. První údaj, který uživatel musí vyplnit, je začátek platnosti licence. Do tohoto políčka se zadává datum, od kdy zakoupená licence má začít platit. Následují položky, která se zobrazují jen nepřihlášeným uživatelům.

Tyto položky slouží k vytvoření uživatelského účtu, na který se naváže zakoupená licence (zadává se email a heslo). Posledními údaji, které se v tomto kroku musí vyplnit, jsou fakturační údaje (kontaktní osoba, telefonní číslo, adresa, případně firemní údaje). V posledním kroku objednávkového formuláře (Příloha 43) se uživateli zobrazí rekapitulace celé objednávky. Uživatel má v tomto kroku zobrazené všechny údaje, které v předchozích krocích zadal. Nachází se zde také kontrola, jestli formulář nevyplnil robot (pomocí Recaptcha od Google). Z tohoto kroku je uživatel přesměrován na platební bránu GoPay, kde postupuje podle zobrazených instrukcí.

3.4.1 Datový model

Diagram níže (Obrázek 21) je zobrazen diagram tříd modulu platební brány. Tento diagram obsahuje vazby na třídy, které byly popsány v diagramu výše v této práce (Obrázek 14).

Obrázek 21 – Diagram tříd modulu platební brány



(Zdroj: Autor)

Třída Order reprezentuje objednávku v systému. Tato třída je namapována na entitu pomocí Doctrine (tudíž její instance jsou ukládány v databázi). Objednávka se tvoří pomocí čtyřkrokového formuláře, který byl popsán výše. Tato třída obsahuje atributy:

- id – Tento atribut je jedinečný identifikátor instance objektu.
- licenseManager – Do tohoto atributu je uložena instance objektu User, která reprezentuje uživatele, kterému bude vytvořena zakoupená licence.
- name – V tomto atributu se eviduje jméno, které se zadává ve třetím kroku objednávkového formuláře (část kontaktních údajů potřebných k fakturaci).
- phoneNumber – Zde se ukládá telefonní číslo (část kontaktních údajů potřebných k fakturaci).
- street – Atribut street obsahuje ulici (část kontaktních údajů potřebných k fakturaci).
- municipality – Do toho atributu je vložena obec (část kontaktních údajů potřebných k fakturaci).
- zip – V tomto atributu se eviduje PSČ obce (část kontaktních údajů potřebných k fakturaci).
- company – Zde se uchovává booleanovská hodnota, která určuje, jestli si uživatel přeje zadat firemní údaje (fakturace proběhne na tuto firmu).
- companyName – Do tohoto pole se uchovává textová hodnota názvu firmy.
- ico – Atribut ico uchovává IČO firmy.
- dic – Tento atribut uchovává DIČ firmy.
- period – Zde se eviduje počet dní, na které je zakoupena licence.
- validityStartDate – Atribut validityStartDate uchovává datum, od kdy začíná licence platit.
- price – Tento atribut obsahuje cenu objednávky.
- modulesGroup – Pokud si uživatel vybral z předvolených balíčků aplikací, je tento balíček evidován zde. Jinak má tento atribut hodnotu NULL.
- ownModulesSelection – V tomto atributu je false, pokud si vybral uživatel z předvolených balíčků aplikací a obsahuje true, pokud si vybral vlastní skladbu modulů.
- modules – Jestliže si uživatel vybral vlastní skladbu modulů, jsou uchovány v této kolekci. Jinak je kolekce prázdná.

Třída Transaction uchovává údaje o probíhající transakci s platební bránou GoPay. Tato třída je velmi důležitá, protože uchovává údaje o tom, v jakém stavu se nachází transakce atd. Třída obsahuje atributy:

- id – Tento atribut je jedinečný identifikátor instance objektu.
- transactionID – Před přesměrování na platební bránu dochází k vytvoření transakce v systému platební brány a tento atribut uchovává její jedinečný identifikátor.
- price – V tomto atributu se eviduje cena, která byla odeslána platební bráně. Platební brána GoPay nepřijímá desetinná místa a místo toho hodnotu 100,50 přijímá jako 10050. Z tohoto důvodu se eviduje cena na dvou místech (jednou v normálním tvaru a jednou v tvaru pro GoPay).
- currency – Zde je uchována hodnota reprezentující měnu, v které transakce probíhá.
- status – V této textové hodnotě je uchován status, v kterém se nachází transakce. Platební brána GoPay průběžně na speciální URL adrese (tato URL je jí předána při založení transakce) říká, že došlo ke změně statusu a tento status je následně uchován i na straně EPIS® Online.
- order – Tímto atributem dochází k propojení na objednávku.
- processed – Tento atribut udává hodnotu, jestli byla vytvořena licence k zaplacené transakci. Po zaplacení platební brána GoPay odešle změnu statusu, že bylo zaplaceno. Tuto změnu však odešle několikrát a licence se musí založit jen jednou. Proto při změně statusu na zaplaceno je vytvořena licence a do tohoto atributu je uloženo true, čímž EPIS® Online má k dispozici informaci, že licence již byla vytvořena.
- viewed – Po dokončení manipulace s platební branou je uživatel přesměrován zpátky na EPIS® Online. Konkrétně je přesměrován na URL adresu, která mu zobrazí, v jakém statusu se nachází jeho objednávka. Pro analytické účely (Google Analytics, Sklik atd.) je při prvním zobrazení statusu (tento atribut eviduje, jestli již status byl zobrazen) zaplacené objednávky do stránky vložen JavaScript, který upozorní analytické služby, že došlo ke koupi produktu.
- created – V tomto atributu se eviduje datum, kdy byla transakce vytvořena.

3.4.2 Tvorba objednávkového formuláře

Tvorba vícekrokového formuláře není defaultně v Symfony implementována, a proto byl v EPIS® Online pro tuto funkcionalitu využit balíček CraueFormFlowBundle. Implementace tohoto balíčku do projektu je velmi jednoduchá. Nejdříve je potřeba vytvořit třídu, která definuje jednotlivé kroky formuláře. V ukázce PHP kódu 22 je vidět třída OrderFlow, která dědí z třídy FormFlow a přepisuje její dvě metody. Metoda loadStepsConfig vrací asociativní pole v předepsaném tvaru, kde jsou definované jednotlivé kroky formuláře, jejich název a který formulář se má v těch krocích použít. Pro všechny kroky objednávkového formuláře je využita stejná třída a to OrderType. Metoda getName vrací název formuláře.

Ukázka PHP kódu 22 - Definice kroků formuláře

```php
namespace EpisBundle\Form\Main\Flow;


use Craue\FormFlowBundle\Form\FormFlow;
use EpisBundle\Form\Main\OrderType;

class OrderFlow extends FormFlow
{
    protected function loadStepsConfig()
    {
        return [
            [
                'label'     => "modules_selection",
                'form_type' => OrderType::class,
            ],
            [
                'label'     => "period_and_number_of_activations_selection",
                'form_type' => OrderType::class,
            ],
            [
                'label'     => 'billing_information',
                'form_type' => OrderType::class,
            ],
            [
                'label'     => 'recapitulation',
                "form_type" => OrderType::class
            ],
        ];
    }

    public function getName()
    {
        return "modules_order";
    }
}
```

(Zdroj: Autor)

Dále je potřeba vytvořit třídu reprezentující objednávkový formulář. Definice třídy OrderType, která k tomuto slouží, je k dispozici v příloze 44. Zde není nic výjimečného od definice normální formuláře. Načítají se zde služby konstruktorem, které jsou následně využity v metodě buildForm a také je zde k vidění v metodě configureOptions nastavení třídy, ke které se formulář vztahuje. Rozdělení formuláře do jednotlivých kroků probíhá až v metodě buildForm této třídy (Ukázka PHP kódu 23). Zde pomocí hodnoty v poli options je možno zjisti, o který krok formuláře jde. V závislosti na to jsou pomocí argumentu builder přidány formulářové prvky.

Ukázka PHP kódu 23 – Metoda builForm třídy OrderType

```php
public function buildForm(FormBuilderInterface $builder, array $options)
{
    if ($options["flow_step"] == 1)
    {
        $builder->add( child: "modulesGroup",  type: EntityType::class, [...]);
        $builder->add( child: "modules",  type: EntityType::class, [...]);
        $builder->add( child: "ownModulesSelection",  type: CheckboxType::class, [...]);
        $builder->addEventListener( eventName: FormEvents::PRE_SET_DATA, function (FormEvent $formEvent) {
            $modulesOder = $formEvent->getData();
            $this->purchaseManager->correctModulesOrder($modulesOder);
            $formEvent->setData($modulesOder);
        });
        return;
    }
    if ($options["flow_step"] == 2)
    {
        $builder->add( child: "period",  type: IntegerType::class, [...]);
        $builder->add( child: "numberOfActivations",  type: IntegerType::class, [...]);
        return;
    }
    if ($options["flow_step"] == 4 && $this->environment == "prod")
    {
        $builder->add( child: "recaptcha",  type: EWZRecaptchaType::class, [...]);
        return;
    }
    if ($this->user == null)
    {
        $builder->add( child: "licenseManager",  type: RegistrationType::class, [...]);
    }
    $builder->add( child: "validityStartDate",  type: DateType::class, [...]);
    $builder->add( child: "name",  type: TextType::class, [...]);
    $builder->add( child: "phoneNumber",  type: TextType::class, [...]);
    $builder->add( child: "street",  type: TextType::class, [...]);
    $builder->add( child: "municipality",  type: TextType::class, [...]);
    $builder->add( child: "zip",  type: TextType::class, [...]);
    $builder->add( child: "company",  type: CheckboxType::class, [...]);
    $builder->add( child: "companyName",  type: TextType::class, [...]);
    $builder->add( child: "ico",  type: TextType::class, [...]);
    $builder->add( child: "dic",  type: TextType::class, [...]);
    $builder->addEventListener( eventName: FormEvents::PRE_SET_DATA, function (FormEvent $formEvent) {
        $order = $formEvent->getData();
        $this->purchaseManager->correctOrder($order);
        $formEvent->setData($order);
    });
}
```

(Zdroj: Autor)

Do výše popsaného formuláře jsou pomocí metody buildForm přidány dvě metody, které naslouchají na odeslání formuláře na server. Tyto metody jsou volány před samotnou validací formuláře. První tato metoda je do formuláře přidána v jeho prvním kroku a řeší situaci, kdy si uživatel navolí vlastní výběr modulů, ale vybere si přesnou skladbu modulů, která se nachází v nějakém předvoleném balíčku. V tomto případě je jeho objednávce nastaveno, že si vybral tento předvolený balíček a je mu nastavena výhodnější cena. Druhá z těchto metod naslouchá ve třetím kroku formuláře a řeší situaci, která by mohla nastat, pokud by formulář využíval útočník. Pokud je nastaveno, že uživatel nechtěl zadat firemní údaje (pomocí atributu company třídy Order) a v atributech companyName, ico a dic jsou nastavené nějaké hodnoty, jsou tyto hodnoty vymazány (hodnoty těchto atributů mají obsahovat hodnotu NULL, pokud uživatel nevybral, že chce zadat firemní údaje).

3.4.3 Akce řadiče

Propojení s platební branou GoPay je velmi jednoduché, jelikož je k dispozici pro Symfony balíček, který řeší veškerou interakci s ní (založení transakce, dotázání na stav transakce atd.). Práci s objednávkami obstarává řadič OrderController. Jeho akce, která zajišťuje interakci a zpracování objednávkového formuláře je k dispozici v příloze 47. Tato akce načítá několik služeb, a to: službu na výpočet ceny produktu (parametr purchaseManager), službu na tvorbu vícekrokových formulářů (parametr flow, což je instance třídy OrderFlow popsané v předchozí kapitole), službu na interakci s platební branou GoPay (parameter paymentsManager) a službu umožňující ukládání entit do databáze (parameter mainObjectManager). Nejdříve je do proměnné order vytvořena instance třídy Order, která reprezentuje objednávku a jejím atributům jsou přiřazeny defaultní hodnoty. Následuje volání metody bind proměnné flow, který si bere za argument proměnnou order. Za proměnnou flow se skrývá vícekrokový formulář popsaný v předchozí kapitloe. Dále je pomocí proměnné flow vytvořen formulář a uložen do proměnné form. Následuje validace tohoto formuláře. Jestliže formulář není validní, uživatel se neposouvá do dalšího kroku formuláře a je mu vykreslen současný stav formuláře společně s validačními chybami. Pokud je však formulář validní, probíhá tělo podmínky, v kterém se tvoří další krok formuláře, za předpokladu, že je to možné. Tento krok je následně předán šabloně, která ho vykreslí. Jestliže se však stav procesu nachází v posledním kroku formuláře a tento krok je validní, proběhne následující akce. Nejdříve je volána metoda reset na proměnné flow. Tím dojde

k vymazání dat ze session (jednotlivé hodnoty, které zadal uživatel v předchozích krocích se ukládají do session). Následuje vytvoření instance objektu Transaction a její uložení do proměnné transaction. Této proměnné jsou nastaveny defaultní hodnoty a je uložena do databáze. Tato proměnná zachycuje stav transakce na straně systému. Následuje vytvoření transakce na straně platební brány pomocí služby paymentsManager metodou createPayment. Této metodě je předáno asociativní pole hodnot (cena, název produktu, návratová URL adresa po dokončení manipulace s platební bránou, URL adresa sloužící komunikující s platební bránou GoPay a ukládající stav transakce na straně systému atd.). Metoda vrátí do proměnné response asociativní pole, které obsahuje hodnoty transakce na straně platební brány. Instance objektu v proměnné transacion si z tohoto pole uloží do svých atributů ID transakce na straně platební brány a současný status transakce. Tím dojde k provázání transakce v systému EPIS® Online a platební brannou. Následně se proměnná transaction uloží do databáze pomocí služby mainObjectManager. Vše končí přesměrováním uživatele na platební bránu (URL adresa platební brány se také čerpá z pole response).

S výše popsanou akcí úzce souvisí akce orderStatusTransaction (Příloha 48). Tuto akci volá platební brána (URL adresa této akce je platební bráně předána při založení transakce) při jakékoliv změně stavu transakce na straně platební brány. Tato akce tento stav uloží do databáze k příslušné instanci objektu Transaction. Tím pádem systém EPIS® Online přesně ví, v jakém stavu se jakákoliv transakce nachází. Pokud platební brána vrátí stav, že platba byla úspěšně provedena, a ještě nebyla objednávce vytvořena licence (toto se určí pomocí atributu processed instance objektu Transaction), je tato licence vytvořena podle údajů v objednávce (je vytvořena aktivace a případně i uživatel). Po vytvoření licence je do atributu processed proměnné transaction uložena hodnota true, což zamezí opětovné vytvoření licence (pokud by platební brána opět volala URL adresu této akce).

Další akci, která souvisí s objednávkou, je orderShowAction (Příloha 46). Na tuto akci je přesměrován uživatel po dokončení interakce s platební bránou. Z URL adresy je pomocí předaného ID načtena transakce, která se k celému procesu vztahuje. Pokud je tato URL adresa navštívena poprvé (toto se určí pomocí atributu viewed instance objektu Transaction), tak je uživateli do odpovědi přidán i JavaScript kód, který o proběhnutém obchodu upozorní analytické služby (Google Analytics, Sklik atd.).

Poslední akci související s objednávkou je modulesOrderPriceAction (Příloha 45). Tato akce je volána z druhého kroku objednávky (Příloha 40). V tomto kroku si uživatel volí počet dní, na které zakupuje licenci (produkt si vybral již v přechozím kroku) a na základě toho je mu zobrazena cena (cena se odvíjí od produktu a počtu dní, na které si zakupuje licenci k tomuto produktu). Při každé změně počtu dní musí být znovu přepočítána výsledná cena, a to je provedeno pomocí AJAX požadavku na tuto akci (tato akce vrací přepočtenou cenu na základě předaných parametrů) (uživateli je přepočtena cena bez nutnosti znovu načíst celou stránku).

4 Závěr

Frameworky pro vývoj jakýchkoliv druhů aplikací (webové, stolní, mobilní atd.) mají v dnešní době nenahraditelné místo. Vývojářům poskytují připravené nástroje řešící různé druhy problémů specifických pro daný typ aplikace, čímž velmi urychlují a usnadňují vývoj. Zvlášť na rychlost je v současnosti kladený velký důraz, jelikož velmi mnoho podnikatelských nápadů je postavených na realizaci prostřednictvím nějaké aplikace a čím dříve bude nápad realizován, tím větší je šance, že na trhu bude první (nebo mezi prvními). Samotné Symfony patří mezi nejlepší frameworky určené k vývoji webových aplikací, což dokazuje jeho stále rostoucí popularita. Je rychlé, lehce adaptovatelné, navržené pomocí nejmodernějších návrhových vzorů, nabízí strmou učící křivku a má mnoho dalších výhod. Nedá se předpokládat, že by v dohledné době z trhu webových frameworků zmizelo.

Praktickým výstupem této diplomové práce je webová aplikace EPIS® Online, která umožňuje uživatelům prohlížení právních předpisů, vyhlášek ministerstev, judikátů a mnoho dalšího. Systém obsahuje intuitivní vyhledávání (jednotlivé prvky vyhledávacích formulářů obsahují našeptávače) a předpřipravené přístupy na záznamy v systému. Uživatelé si mohou jednotlivé záznamy ukládat do oblíbených, psát k nim komentáře a následně k těmto záznamům lehce přistupovat. Dále mohou porovnávat jednotlivé znění právních předpisů a zobrazit si rozdíly jednotlivých verzí. Aplikace byla vyvinuta pomocí Symfony a osvědčených postupů, které se pojí s tímto frameworkem, tudíž je lehce rozvinutelná o další moduly. Obchodní model systému byl navržen tak, aby si uživatel z EPIS® Online mohl vybrat pouze části, které potřebuje a nemusel platit za části systému, které by nevyužil. Dle tohoto výběru se mu náležitě upraví cena. Ovládání celé aplikace bylo navrženo s citem tak, aby bylo co nejvíce intuitivní a zorientovali se v něm i méně technicky zdatní uživatelé. Důraz byl také kladen na responzivitu celé aplikace, která je schopna se přizpůsobit jakémukoliv zařízení a uživatel nepřijde o žádné funkcionality. Aplikace se vyvíjela necelé dva roky a většinu jejího zdrojového kódu napsal autor této práce, ale přesto obsahuje funkcionality, které konkurence nenabízí a díky nim získává EPIS® Online v určitých směrech konkurenční výhodu.

I Summary and keywords

The first part of the thesis is about web applications, MVC architecture and web frameworks. Their role in application development is discussed and the advantages and disadvantages of their use are outlined. In the following part, the Symfony web framework 3.4 is discussed further. Specifically, this section develops its core components and functionality that it uses (structure, configuration files, controllers, entities, templates, models, security, etc.)

The second part of the thesis is dedicated to the development of economic-legal information system EPIS® Online, which was created by Symfony 3.4. Its three modules (security and user license structure, legislation and payment gateway) are described further. The specific processes in the system, actions of controllers, templates, services and many other components are explained in this part.

Key words: economic-legal information system, legal regulations, web application, Symfony, development

II Seznam použitých zdrojů

Armand, S. (2014). Extending Symfony2 Web Application Framework. UK: Packt.

Čápka, D. (2019). Úvod do PHP a webových aplikací [Online]. Retrieved February 07, 2019, from https://www.itnetwork.cz/php/zaklady/php-tutorial-uvod-do-webovych-aplikaci

Čápka, D. (2019). Skládání stránek v PHP [Online]. Retrieved February 12, 2019, from https://www.itnetwork.cz/php/zaklady/php-tutorial-zaklady-skladani-stranek

Elizabeth, L. (2019). Should You Develop a Desktop or Web App? [Online]. Retrieved February 07, 2019, from https://www.sitepoint.com/web-desktop-apps/

Elmbland, S. (2019). How Is Desktop Software Different From an App? [Online]. Retrieved February 07, 2019, from https://www.thebalance.com/what-is-desktop-software-1293673

Christensson, P. (2019). Framework Definition [Online]. Retrieved February 08, 2019, from https://techterms.com/definition/framework

Objektově relační mapování. (2019). Objektově relační mapování [Online]. Retrieved February 10, 2019, from https://cs.wikipedia.org/wiki/Objektov%C4%9B_rela%C4%8Dn%C3%AD_mapov%C3%A1n%C3%AD

Salehi, S. (2016). Mastering Symfony. Birmingham UK: Packt.

Symfony. (2019). Symfony [Online]. Retrieved February 09, 2019, from https://symfony.com/

Technologies market share: Web Application Frameworks. (2019). Technologies market share: Web Application Frameworks [Online]. Retrieved February 08, 2019, from https://www.similartech.com/categories/framework

Twig. (2019). Twig [Online]. Retrieved February 12, 2019, from https://twig.symfony.com/

Web framework. (2019). Web framework [Online]. Retrieved February 08, 2019, from https://en.wikipedia.org/wiki/Web_framework

III Seznam obrázků

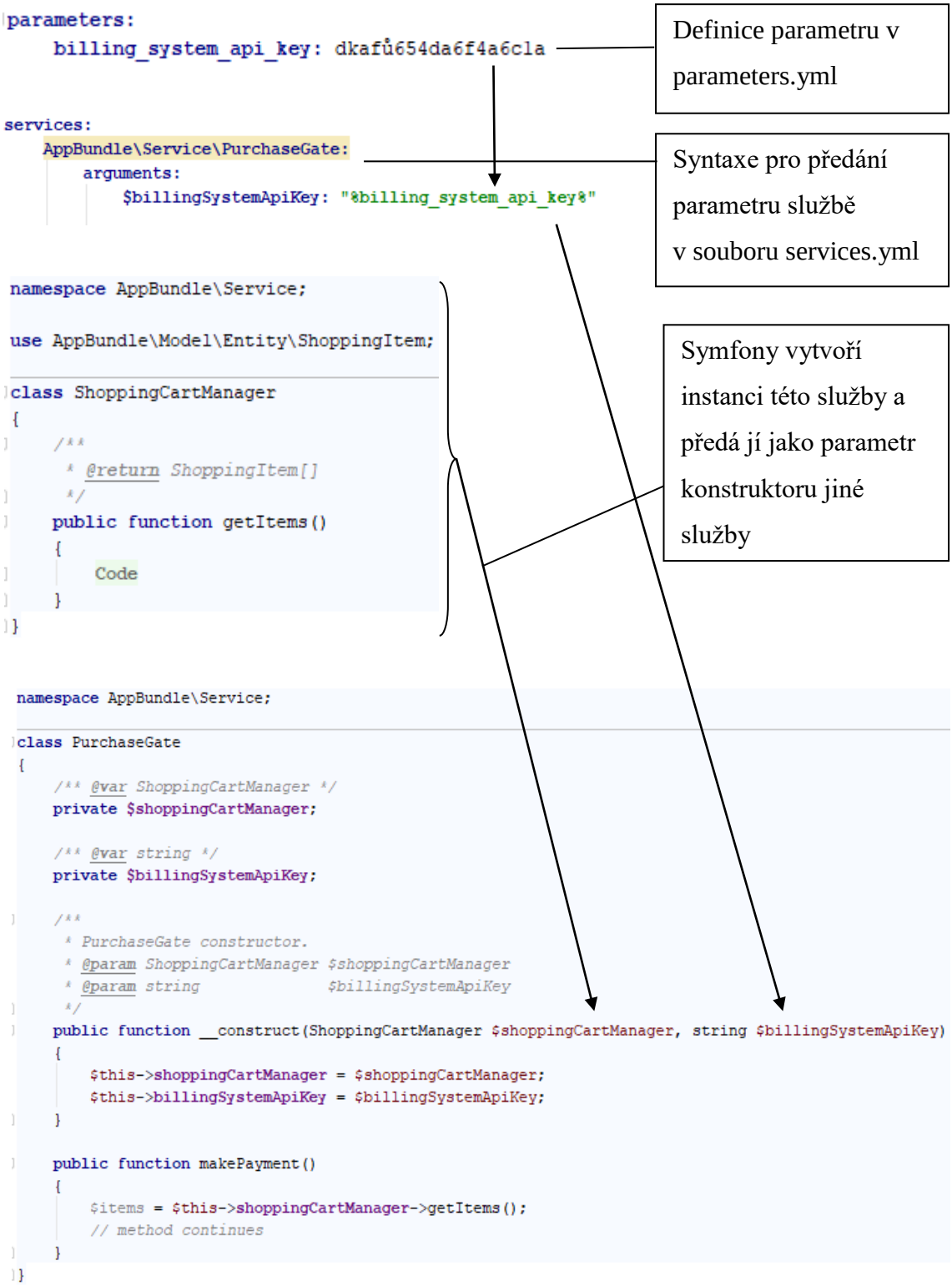
IV Seznam tabulek

V Seznam ukázek PHP kódů

VI Seznam příloh

VII Přílohy

Příloha 1 – Schéma Dependency Injection



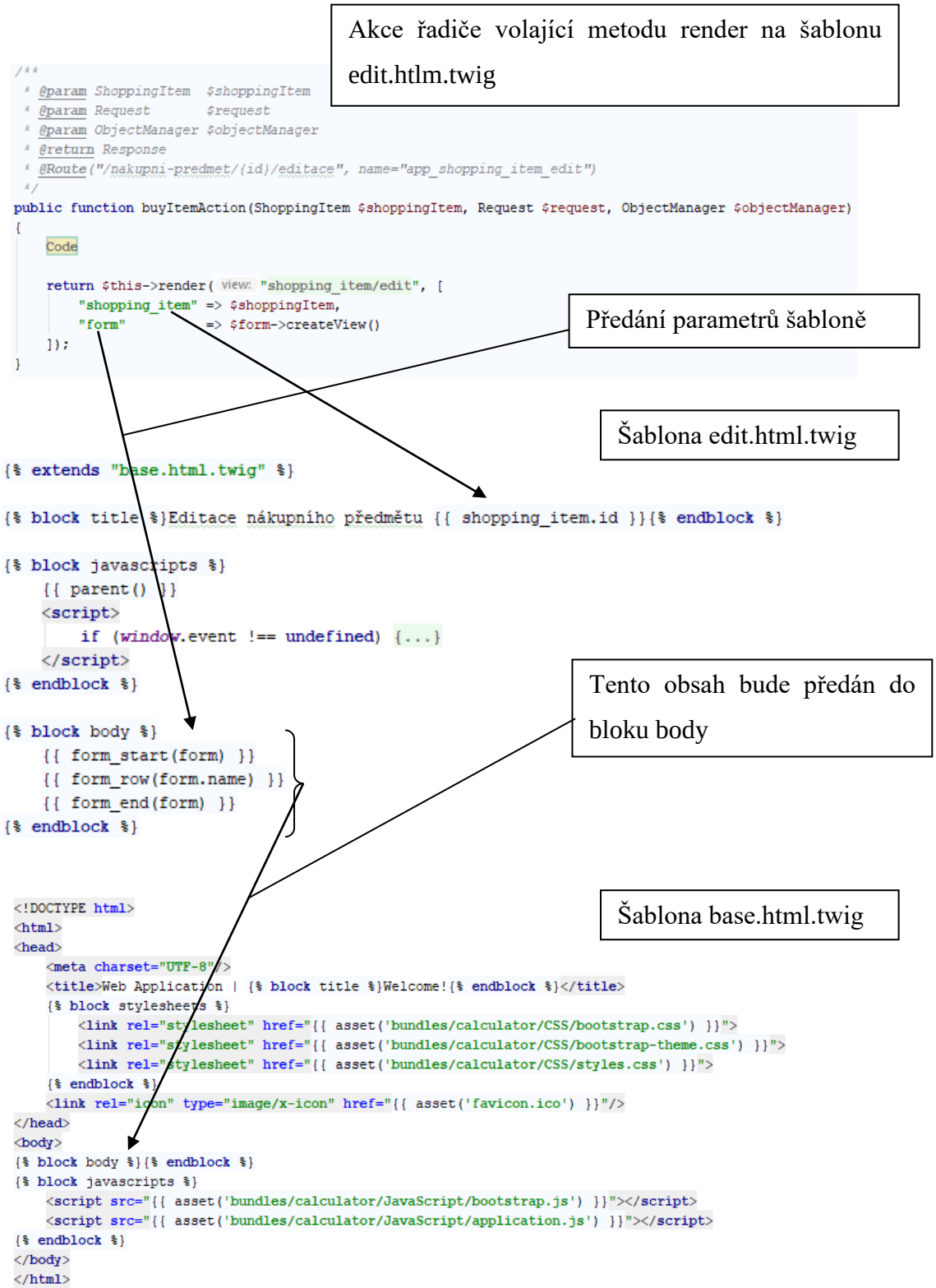
(Zdroj: Autor)

Příloha 2 – Využití formuláře v akci řadiče

```php
namespace AppBundle\Controller;

use AppBundle\Form\ShoppintItemType;
use AppBundle\Service\PurchaseGate;
use AppBundle\Model\Entity\ShoppingItem;
use Doctrine\Common\Persistence\ObjectManager;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

class ShoppingItemController extends Controller
{
    /**
     * @param ShoppingItem  $shoppingItem
     * @param Request       $request
     * @param ObjectManager $objectManager
     * @return Response
     * @Route("/nakupni-predmet/{id}/editace", name="app_shopping_item_edit")
     */
    public function buyItemAction(ShoppingItem $shoppingItem, Request $request, ObjectManager $objectManager)
    {
        $form = $this->createForm( type: ShoppintItemType::class, $shoppingItem, ["validation_groups" => ["administration_edit"]]);
        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid())
        {
            $objectManager->persist($shoppingItem);
            $objectManager->flush();

            $this->addFlash( type: "success",  message: "Nákupní předmět byl editován.");

            return $this->redirectToRoute( route: "homepage");
        }

        return $this->render( view: "shopping_item/edit.html.twig", [
            "shopping_item" => $shoppingItem,
            "form"          => $form->createView()
        ]);
    }
```

(Zdroj: Autor)

Příloha 3 – Skládání šablon a předávání parametrů šablonám v Twig a Symfony



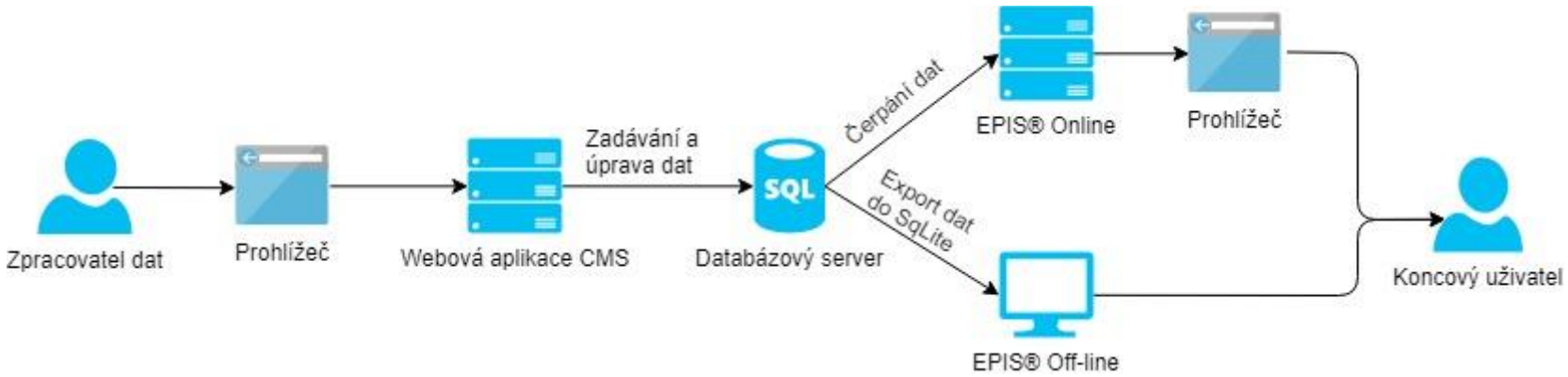

```php
/**
 * @param ShoppingItem  $shoppingItem
 * @param Request       $request
 * @param ObjectManager $objectManager
 * @return Response
 * @Route("/nakupni-predmet/{id}/editace", name="app_shopping_item_edit")
 */
public function buyItemAction(ShoppingItem $shoppingItem, Request $request, ObjectManager $objectManager)
{
    Code

    return $this->render( view: "shopping_item/edit", [
        "shopping_item" => $shoppingItem,
        "form"          => $form->createView()
    ]);
}
```

```twig
{% extends "base.html.twig" %}

{% block title %}Editace nákupního předmětu {{ shopping_item.id }}{% endblock %}

{% block javascripts %}
    {{ parent() }}
    <script>
        if (window.event !== undefined) {...}
    </script>
{% endblock %}

{% block body %}
    {{ form_start(form) }}
    {{ form_row(form.name) }}
    {{ form_end(form) }}
{% endblock %}
```

```twig
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8" />
    <title>Web Application | {% block title %}Welcome!{% endblock %}</title>
    {% block stylesheets %}
        <link rel="stylesheet" href="{{ asset('bundles/calculator/CSS/bootstrap.css') }}">
        <link rel="stylesheet" href="{{ asset('bundles/calculator/CSS/bootstrap-theme.css') }}">
        <link rel="stylesheet" href="{{ asset('bundles/calculator/CSS/styles.css') }}">
    {% endblock %}
    <link rel="icon" type="image/x-icon" href="{{ asset('favicon.ico') }}" />
</head>
<body>
{% block body %}{% endblock %}
{% block javascripts %}
    <script src="{{ asset('bundles/calculator/JavaScript/bootstrap.js') }}"></script>
    <script src="{{ asset('bundles/calculator/JavaScript/application.js') }}"></script>
{% endblock %}
</body>
</html>
```

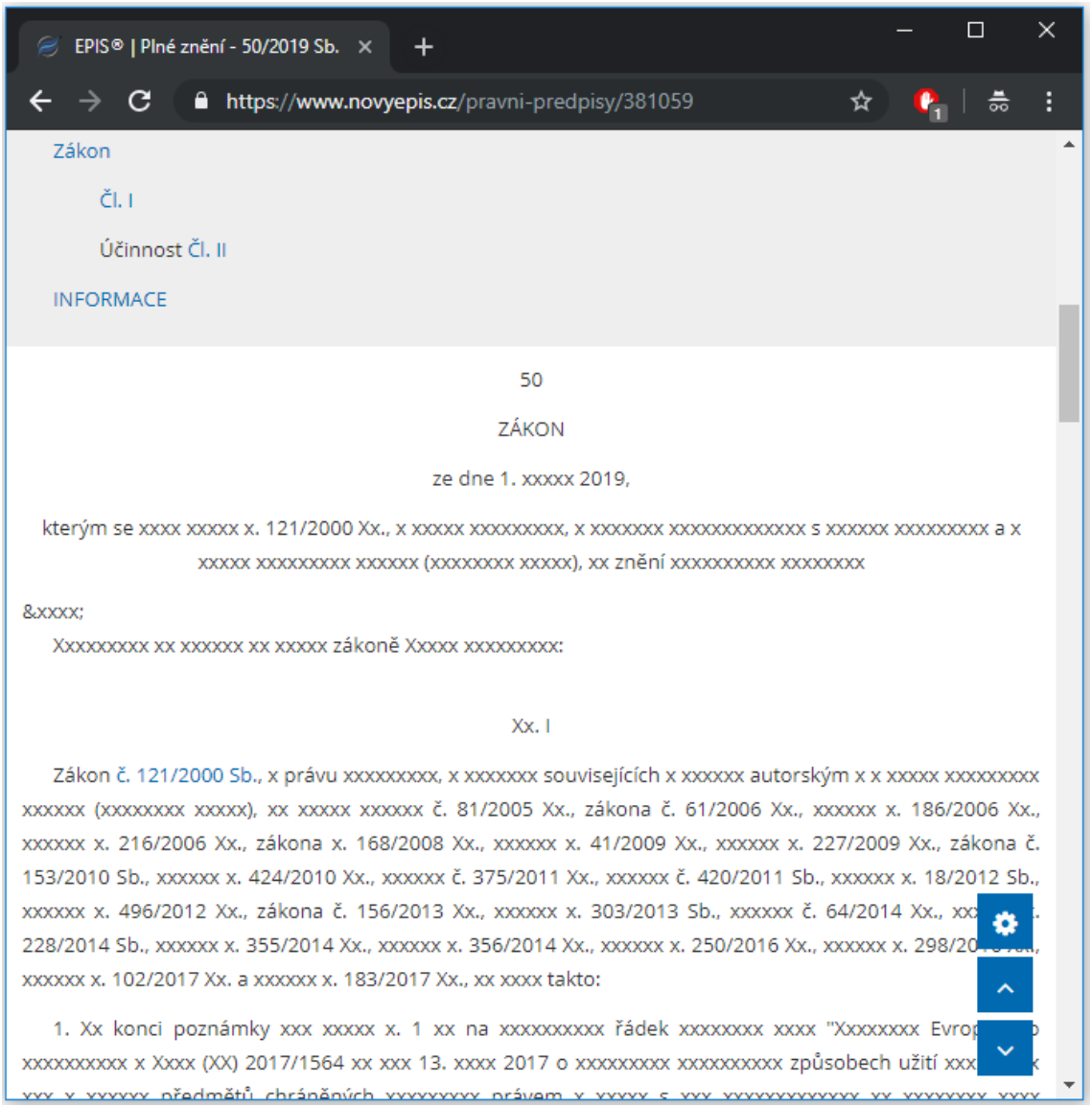
(Zdroj: Autor)

Příloha 4 – Pohyb zpracovaných dat ve firmě GRAND s.r.o.



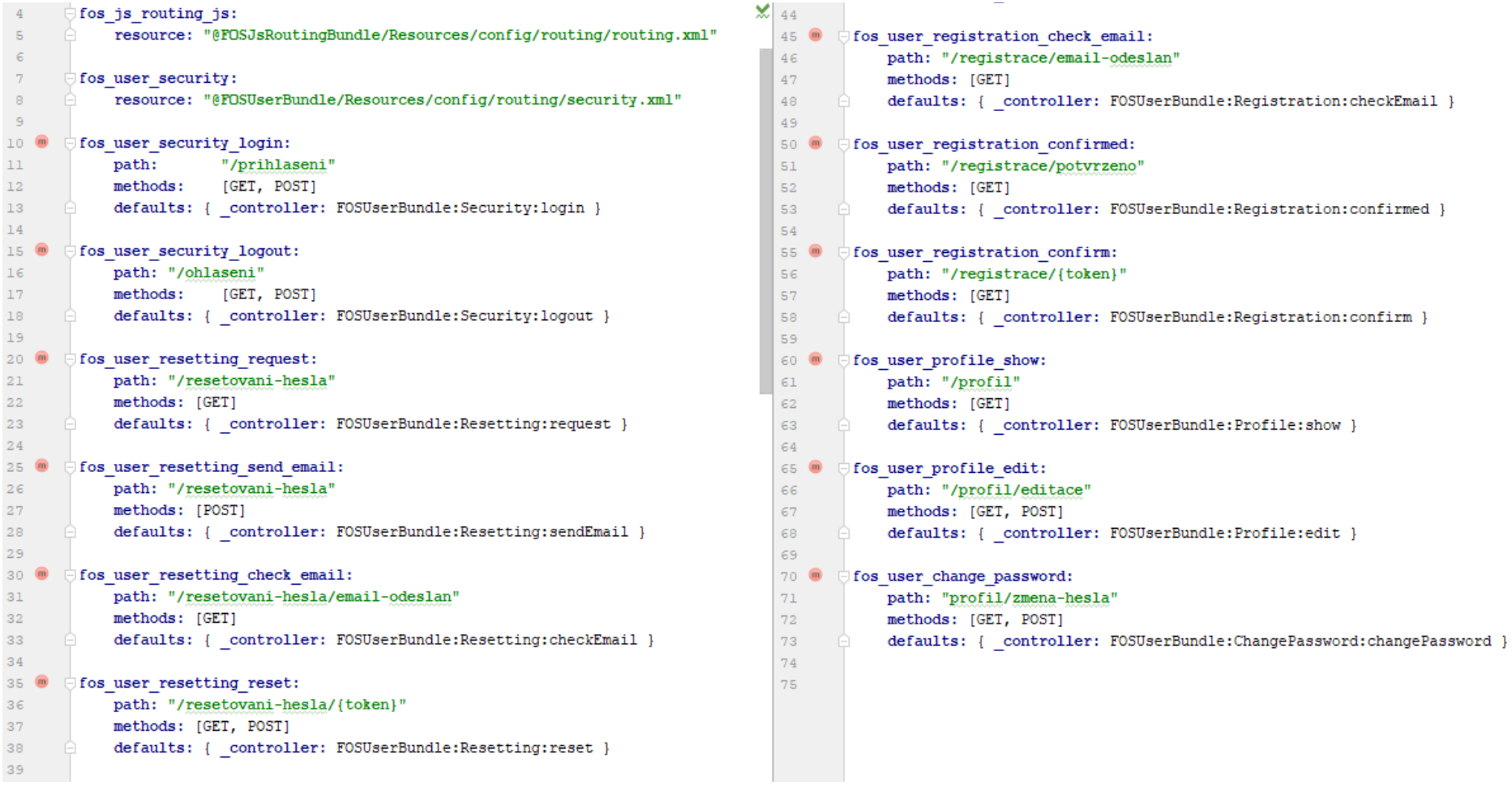
(Zdroj: Autor)

Příloha 5 – Ukázka skrytí textů pro neautorizované uživatele



(Zdroj: https://novyepis.cz)

Příloha 6 – Přepsání adres akcí řadičů FosUserBundle v souboru routing.yml



```yaml
fos_js_routing_js:
    resource: "@FOSJsRoutingBundle/Resources/config/routing/routing.xml"

fos_user_security:
    resource: "@FOSUserBundle/Resources/config/routing/security.xml"

fos_user_security_login:
    path:       "/prihlaseni"
    methods:    [GET, POST]
    defaults: { _controller: FOSUserBundle:Security:login }

fos_user_security_logout:
    path: "/ohlaseni"
    methods:    [GET, POST]
    defaults: { _controller: FOSUserBundle:Security:logout }

fos_user_resetting_request:
    path: "/resetovani-hesla"
    methods: [GET]
    defaults: { _controller: FOSUserBundle:Resetting:request }

fos_user_resetting_send_email:
    path: "/resetovani-hesla"
    methods: [POST]
    defaults: { _controller: FOSUserBundle:Resetting:sendEmail }

fos_user_resetting_check_email:
    path: "/resetovani-hesla/email-odeslan"
    methods: [GET]
    defaults: { _controller: FOSUserBundle:Resetting:checkEmail }

fos_user_resetting_reset:
    path: "/resetovani-hesla/{token}"
    methods: [GET, POST]
    defaults: { _controller: FOSUserBundle:Resetting:reset }

fos_user_registration_check_email:
    path: "/registrace/email-odeslan"
    methods: [GET]
    defaults: { _controller: FOSUserBundle:Registration:checkEmail }

fos_user_registration_confirmed:
    path: "/registrace/potvrzeno"
    methods: [GET]
    defaults: { _controller: FOSUserBundle:Registration:confirmed }

fos_user_registration_confirm:
    path: "/registrace/{token}"
    methods: [GET]
    defaults: { _controller: FOSUserBundle:Registration:confirm }

fos_user_profile_show:
    path: "/profil"
    methods: [GET]
    defaults: { _controller: FOSUserBundle:Profile:show }

fos_user_profile_edit:
    path: "/profil/editace"
    methods: [GET, POST]
    defaults: { _controller: FOSUserBundle:Profile:edit }

fos_user_change_password:
    path: "profil/zmena-hesla"
    methods: [GET, POST]
    defaults: { _controller: FOSUserBundle:ChangePassword:changePassword }
```

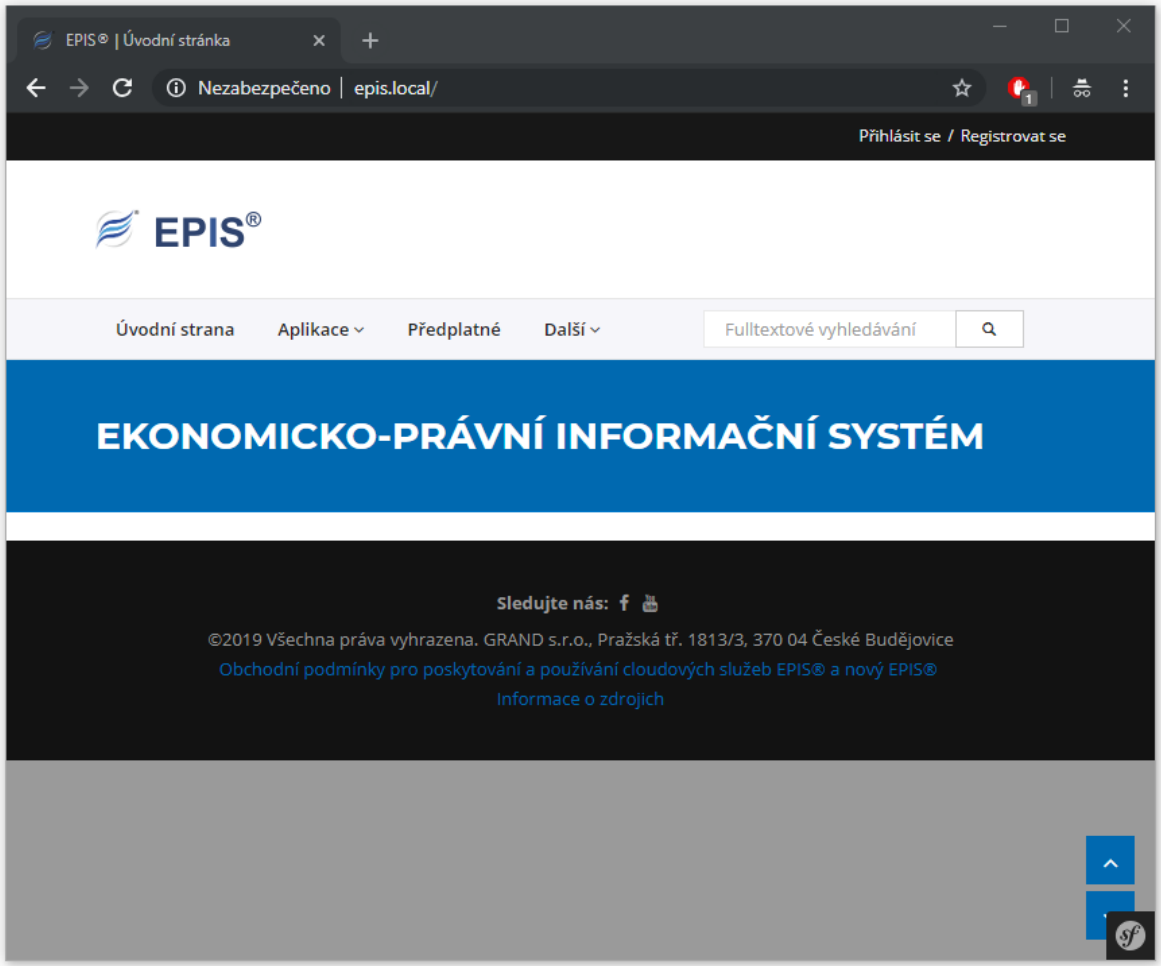
(Zdroj: Autor)

Příloha 7 – Deklarace a konstruktor třídy KernelRequestSubscriber

```php
namespace EpisBundle\EventListener;

use EpisBundle\Entity\Main\User;
use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use Symfony\Component\HttpFoundation\RedirectResponse;
use Symfony\Component\HttpFoundation\Session\SessionInterface;
use Symfony\Component\HttpKernel\Event\GetResponseEvent;
use Symfony\Component\HttpKernel\KernelEvents;
use Symfony\Component\Routing\Generator\UrlGeneratorInterface;
use Symfony\Component\Security\Core\Authentication\Token\Storage\TokenStorageInterface;
use Symfony\Component\Security\Core\Authorization\AuthorizationCheckerInterface;
use Symfony\Component\Security\Core\Exception\AuthenticationCredentialsNotFoundException;

class KernelRequestSubscriber implements EventSubscriberInterface
{

    /** @var TokenStorageInterface */
    private $tokenStorage;

    /** @var AuthorizationCheckerInterface */
    private $authorizationChecker;

    /** @var SessionInterface */
    private $session;

    /** @var UrlGeneratorInterface */
    private $urlGenerator;

    /**
     * KernelRequestSubscriber constructor.
     * @param TokenStorageInterface         $tokenStorage
     * @param AuthorizationCheckerInterface $authorizationChecker
     * @param SessionInterface              $session
     */
    public function __construct(TokenStorageInterface $tokenStorage, AuthorizationCheckerInterface $authorizationChecker, SessionInterface $session)
    {
        $this->tokenStorage = $tokenStorage;
        $this->authorizationChecker = $authorizationChecker;
        $this->session = $session;
    }
```

(Zdroj: Autor)

Příloha 8 – Hlavní šablona aplikace



(Zdroj: https://novyepis.cz)

Příloha 9 – Twig šablona pro přihlašovací formulář

```twig
{% extends "EpisBundle::base.html.twig" %}

{% trans_default_domain 'FOSUserBundle' %}

{% block title %}Přihlášení{% endblock %}

{% block body %}
    <div class="container">
        <div class="col-md-6 col-md-offset-3">
            <h2 class="mb15">Přihlášení</h2>
            <form role="form" action="{{ path("fos_user_security_check") }}" method="post">
                {% if error %}
                    <div class="row mr0 ml0">
                        <div class="alert alert-danger alert-dismissable">
                            <button type="button" class="close" data-dismiss="alert" aria-hidden="true">×</button>
                            <span class="alert-icon"><i class="fa fa-warning"></i></span><strong>Pozor!</strong>
                            {{ error.messageKey|trans(error.messageData, 'security') }}
                        </div>
                    </div>
                {% endif %}
                <input type="hidden" name="_csrf_token" value="{{ csrf_token }}"/>
                <div class="form-group">
                    <div class="input-group">
                        <span class="input-group-addon"><i class="fa fa-user"></i></span>
                        <input type="text" class="form-control" id="username" name="_username"
                               required="required" value="{{ last_username }}"
                               placeholder="{{ 'form.email'|trans }}">
                    </div>
                </div><!-- End .from-group -->
                </div><!-- End .from-group -->
                <div class="form-group row">
                    <div class="col-xs-6">
                        <div class="checkbox mt0 mb0">
                            <label class="custom-checkbox-wrapper">
                            <span class="custom-checkbox-container">
                                <input type="checkbox" id="remember_me" name="_remember_me"
                                       checked="checked"
                                       value="on"/>
                                <span class="custom-checkbox-icon"></span>
                            </span>
                                <span class="custom-checkbox-text">{{ 'security.login.remember_me'|trans }}</span>
                            </label>
                        </div>
                    </div><!-- End .col-xs-6 -->
                    <div class="col-xs-6 text-right">
                        <a href="{{ path('fos_user_resetting_request') }}">Zapomněli jste heslo?</a>
                    </div><!-- End .col-xs-6 -->
                </div><!-- End .row -->

                <div class="form-group">
                    <button type="submit" class="btn btn-primary min-width">
                        <i class="fa fa-sign-in"></i>{{ 'security.login.submit'|trans }}
                    </button>
                </div>
            </form>
        </div>
    </div>
{% endblock %}
```

(Zdroj: Autor)

Příloha 10 – Twig šablona pro registrační formulář

```
{% extends "@Epis/base.html.twig" %}

{% trans_default_domain 'FOSUserBundle' %}

{% block title %}Registrace{% endblock %}

{% block body %}
    <div class="container">
        <div class="row">
            <div class="col-md-6 col-md-offset-3">
                {{ form_start(form, {'method': 'post', 'action': path('fos_user_registration_register')}) }}
                {{ form_row(form.email) }}
                {{ form_row(form.plainPassword) }}
                {% if app.environment == "prod" %}
                    {{ form_row(form.recaptcha) }}
                {% endif %}
                <div class="form-group">
                    <button class="btn btn-primary min-width" type="submit">
                        <i class="fa fa-sign-in"></i> {{ 'registration.submit'|trans }}
                    </button>
                </div>
                {{ form_end(form) }}
            </div>
        </div>
    </div>
{% endblock %}
```

(Zdroj: Autor)

Příloha 11 – Twig šablona pro vykreslení stránky oznamující vyřízení registrace

```
{% extends "@Epis/base.html.twig" %}

{% trans_default_domain 'FOSUserBundle' %}

{% block title %}Žádost o registraci vyřízena{% endblock %}

{% block body %}
    <div class="container">
        <p>
            {{ 'registration.check_email'|trans({'%email%': user.email}) }}
        </p>
    </div>
{% endblock %}
```

(Zdroj: Autor)

Příloha 12 – Twig šablona pro oznámení o dokončení registrace

```
{% extends "@Epis/base.html.twig" %}

{% trans_default_domain 'FOSUserBundle' %}

{% block title %}Registrace dokončena{% endblock %}

{% block body %}
    <div class="container">
        <p>{{ 'registration.confirmed'|trans({'%username%': user.username}) }}</p>
    </div>
{% endblock %}
```

(Zdroj: Autor)

Příloha 13 – Twig šablona pro profil uživatele

```
{% extends "@Epis/base.html.twig" %}

{% trans_default_domain 'FOSUserBundle' %}

{% block title %}Profil{% endblock %}

{% block body %}
    <div class="container">
        <table class="simple-list mb15">
            <tr>
                <td><strong>ID</strong></td>
                <td>{{ user.id }}</td>
            </tr>
            <tr>
                <td><strong>{{ 'profile.show.email'|trans }}</strong></td>
                <td>{{ user.email }}</td>
            </tr>
        </table>
        <div class="dropdown">
            <button class="btn btn-default dropdown-toggle" type="button" id="dropdown_menu_profile_actions"
                    data-toggle="dropdown" aria-haspopup="true" aria-expanded="true">
                Další nastavení a možnosti  <span class="caret"></span>
            </button>
            <ul class="dropdown-menu" aria-labelledby="dropdown_menu_profile_actions">
                <li><a href="{{ path('fos_user_profile_edit') }}">Editovat profil</a></li>
                <li><a href="{{ path('fos_user_change_password') }}">Změnit heslo</a></li>
            </ul>
        </div>
    </div>
{% endblock %}
```

(Zdroj: Autor)

Příloha 14 – Twig šablona pro formulář pro editaci údajů uživatele

```
{% extends "@Epis/base.html.twig" %}

{% trans_default_domain 'FOSUserBundle' %}

{% block title %}Editace profilu{% endblock %}

{% block body %}
    <div class="container">
        {{ form_start(form, { 'action': path('fos_user_profile_edit') }) }}
        {{ form_row(form.email) }}
        <div class="form-group">
            <button class="btn btn-primary min-width">
                <i class="fa fa-save"></i> {{ 'profile.edit.submit'|trans }}
            </button>
        </div>
        {{ form_end(form) }}
    </div>
{% endblock %}
```

(Zdroj: Autor)

Příloha 15 – Twig šablona pro formulář umožňující změnu hesla

```
{% extends "@Epis/base.html.twig" %}

{% trans_default_domain 'FOSUserBundle' %}

{% block title %}Změna hesla{% endblock %}

{% block body %}
    <div class="container">
        {{ form_start(form, { 'action': path('fos_user_change_password') }) }}
        {{ form_row(form.current_password) }}
        {{ form_row(form.plainPassword) }}
        <div class="form-group">
            <button class="btn btn-primary min-width">
                <i class="fa fa-lock"></i> {{ 'change_password.submit'|trans }}
            </button>
        </div>
        {{ form_end(form) }}
    </div>
{% endblock %}
```

(Zdroj: Autor)

Příloha 16 – Twig šablona pro formulář se žádostí o změnu zapomenutého hesla

```
{% extends "@Epis/base.html.twig" %}

{% import "Macro/_modals" as modals %}

{% trans_default_domain 'FOSUserBundle' %}

{% block title %}Žádost o změnu hesla{% endblock %}

{% block body %}
    <div class="container">
        <div class="row">
            <div class="col-md-6 col-md-offset-3">
                <h1 class="h4">Žádost o změnu hesla</h1>
                <form action="{{ path('fos_user_resetting_send_email') }}" method="POST">
                    <div class="form-group">
                        <div class="input-group">
                            <span class="input-group-addon"><i class="fa fa-user"></i></span>
                            <input type="text" class="form-control" id="username" name="username"
                                   required="required" placeholder="{{ 'form.email'|trans }}">
                        </div>
                    </div>
                    {% set modal_id = random_html_element_id('modal') %}
                    <div class="form-group">
                        <button class="btn btn-primary min-width" type="button" data-toggle="modal" data-target="#{{ modal_id }}">
                            <i class="fa fa-unlock"></i> {{ 'resetting.request.submit'|trans }}
                        </button>
                    </div>
                    {{ modals.submit_modal("Žádost o změnu hesla", "Opravdu chcete odeslat žádost o změnu hesla?", modal_id) }}
                </form>
            </div>
        </div>
    </div>
{% endblock %}
```

(Zdroj: Autor)

Příloha 17 – Twig šablona pro oznámení o vyřízení žádosti o změnu zapomenutého hesla

```
{% extends "@Epis/base.html.twig" %}

{% trans_default_domain 'FOSUserBundle' %}

{% block title %}Žádost o změnu hesla vyřízena{% endblock %}

{% block body %}
    <div class="container">
        <h1 class="h4">Žádost o změnu hesla vyřízena</h1>
        <p>
            {{ 'resetting.check_email'|trans({'%tokenLifetime%': tokenLifetime})|nl2br }}
        </p>
    </div>
{% endblock %}
```

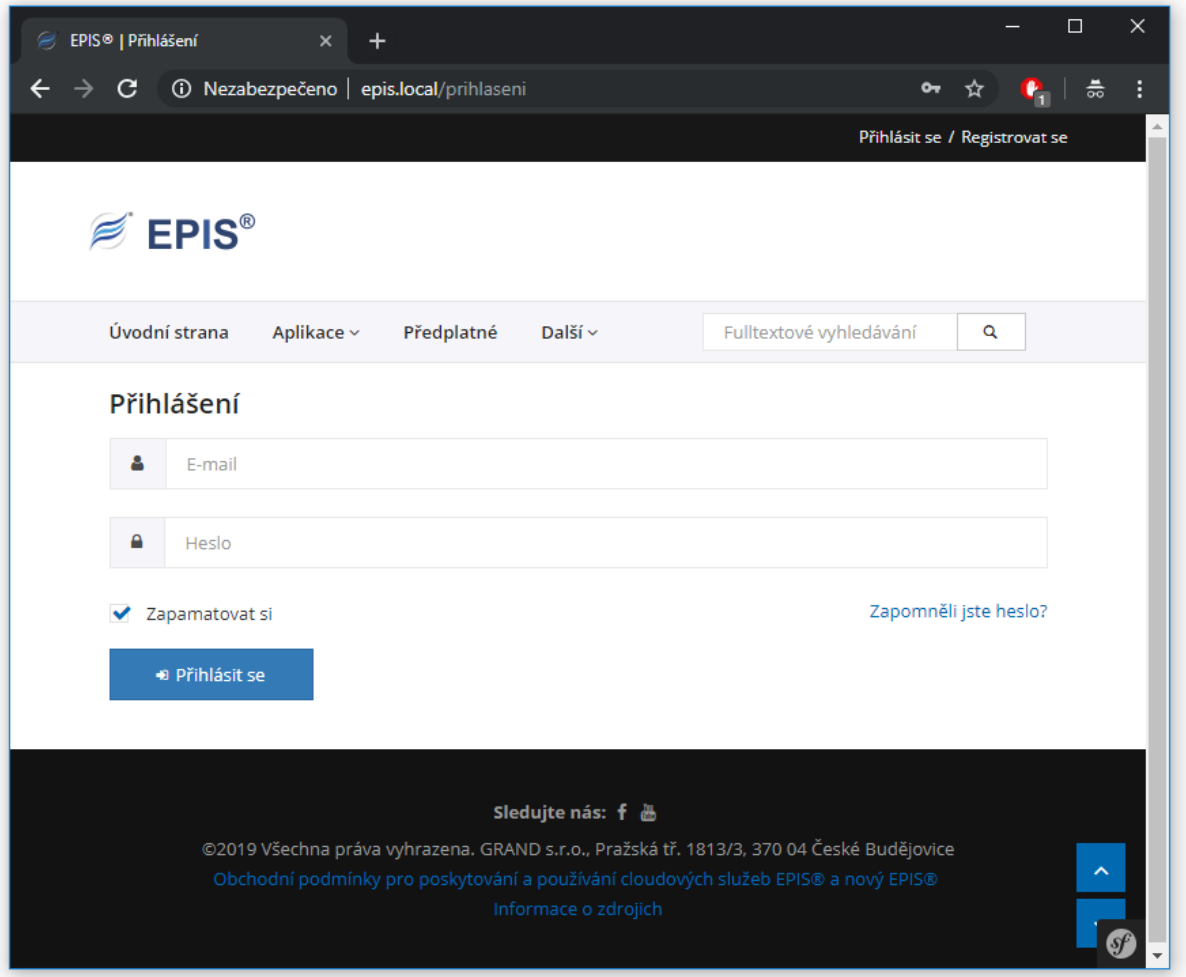
(Zdroj: Autor)

Příloha 18 – Twig šablona pro formulář umožňující změnu zapomenutého hesla

```
{% extends "@Epis/base.html.twig" %}

{% import "Macro/_modals" as modals %}

{% trans_default_domain 'FOSUserBundle' %}

{% block title %}Změna hesla{% endblock %}

{% block body %}
    <div class="container">
        <div class="row">
            <div class="col-md-6 col-md-offset-3">
                <h1 class="h4">Změna hesla</h1>
                {{ form_start(form, { 'action': path('fos_user_resetting_reset', {'token': token}) }) }}
                {{ form_widget(form) }}
                <div class="form-group">
                    {% set modal_id = random_html_element_id('modal') %}
                    <button type="button" class="btn btn-primary min-width" data-toggle="modal" data-target="#{{ modal_id }}">
                        <i class="fa fa-unlock"></i> {{ 'resetting.reset.submit'|trans }}
                    </button>
                    {{ modals.submit_modal("Změna hesla", "Opravdu chcete změnit heslo?", modal_id) }}
                </div>
                {{ form_end(form) }}
            </div>
        </div>
    </div>
{% endblock %}
```

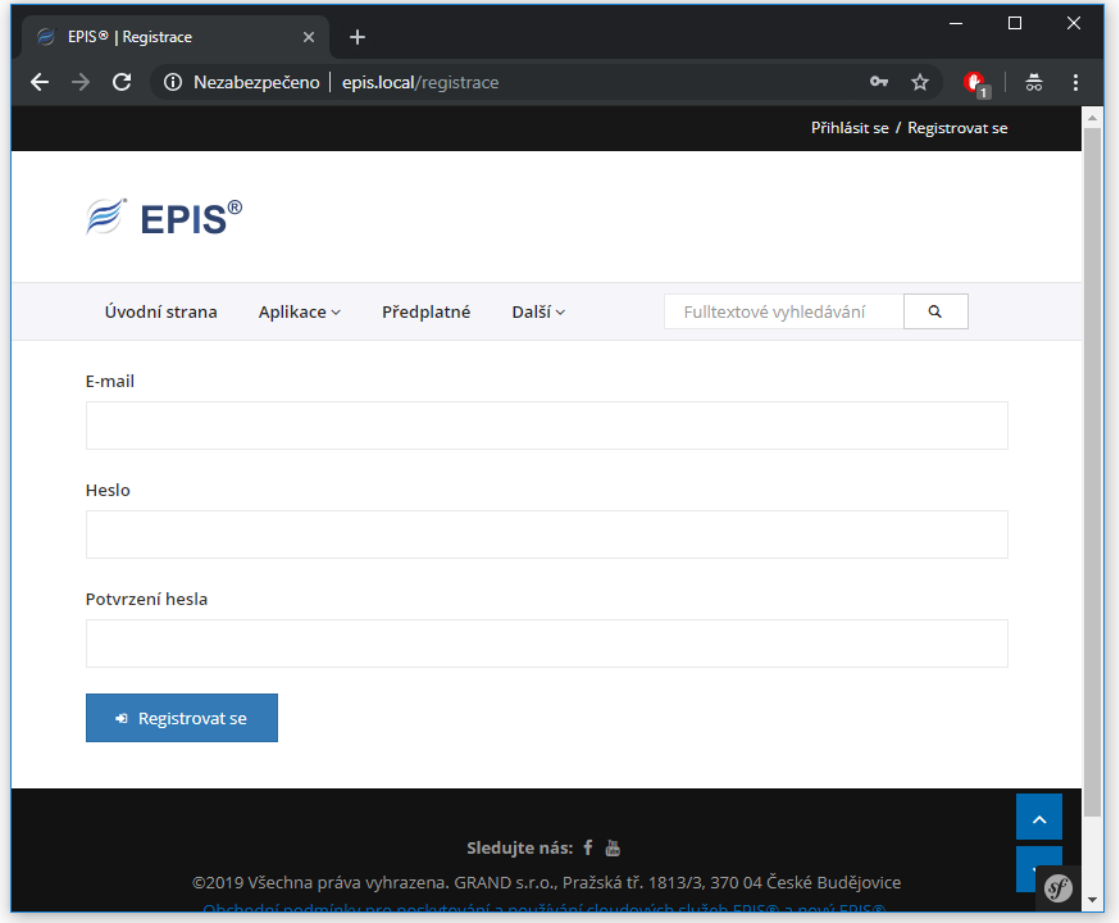
(Zdroj: Autor)

Příloha 19 – Přihlašovací obrazovka aplikace



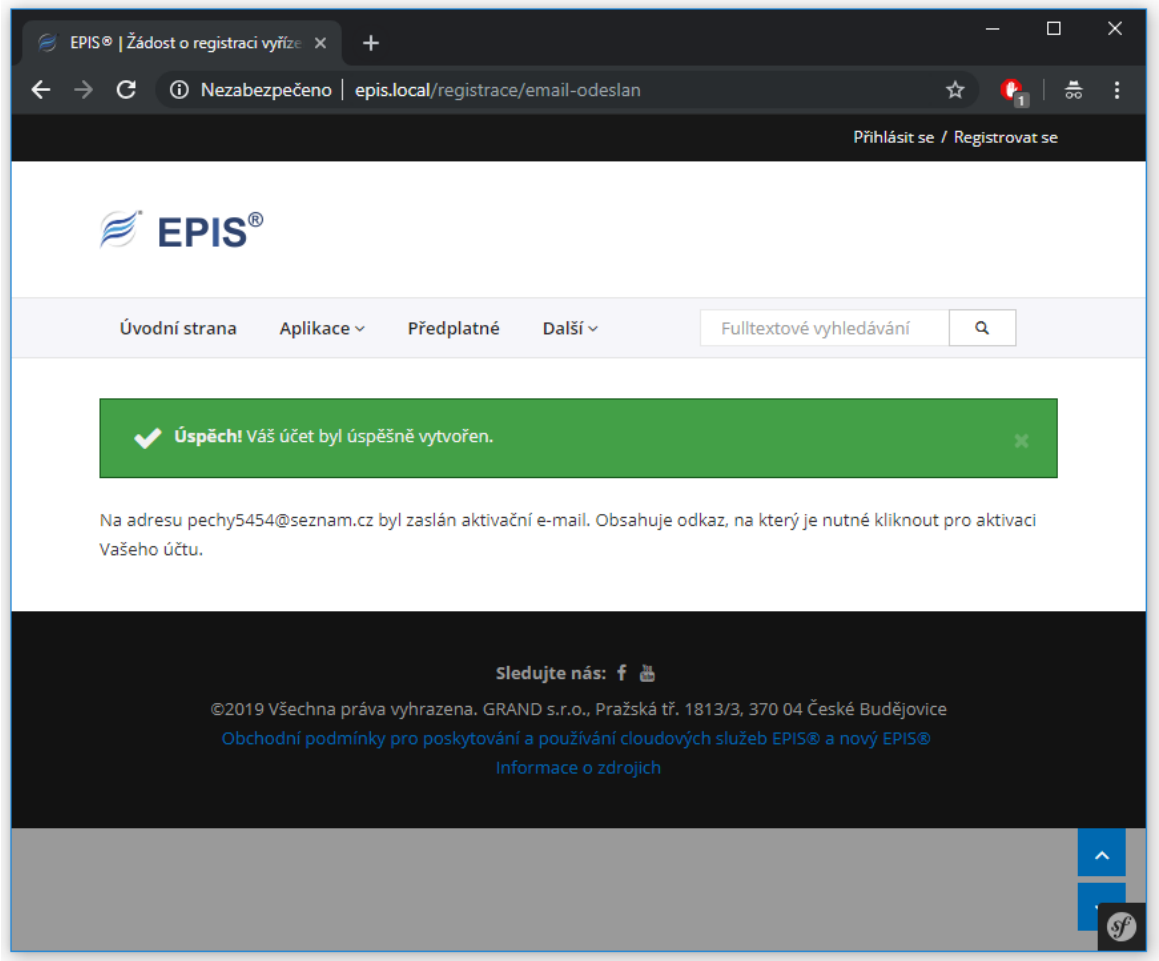
(Zdroj: https://novyepis.cz)

Příloha 20 – Registrační obrazovka aplikace



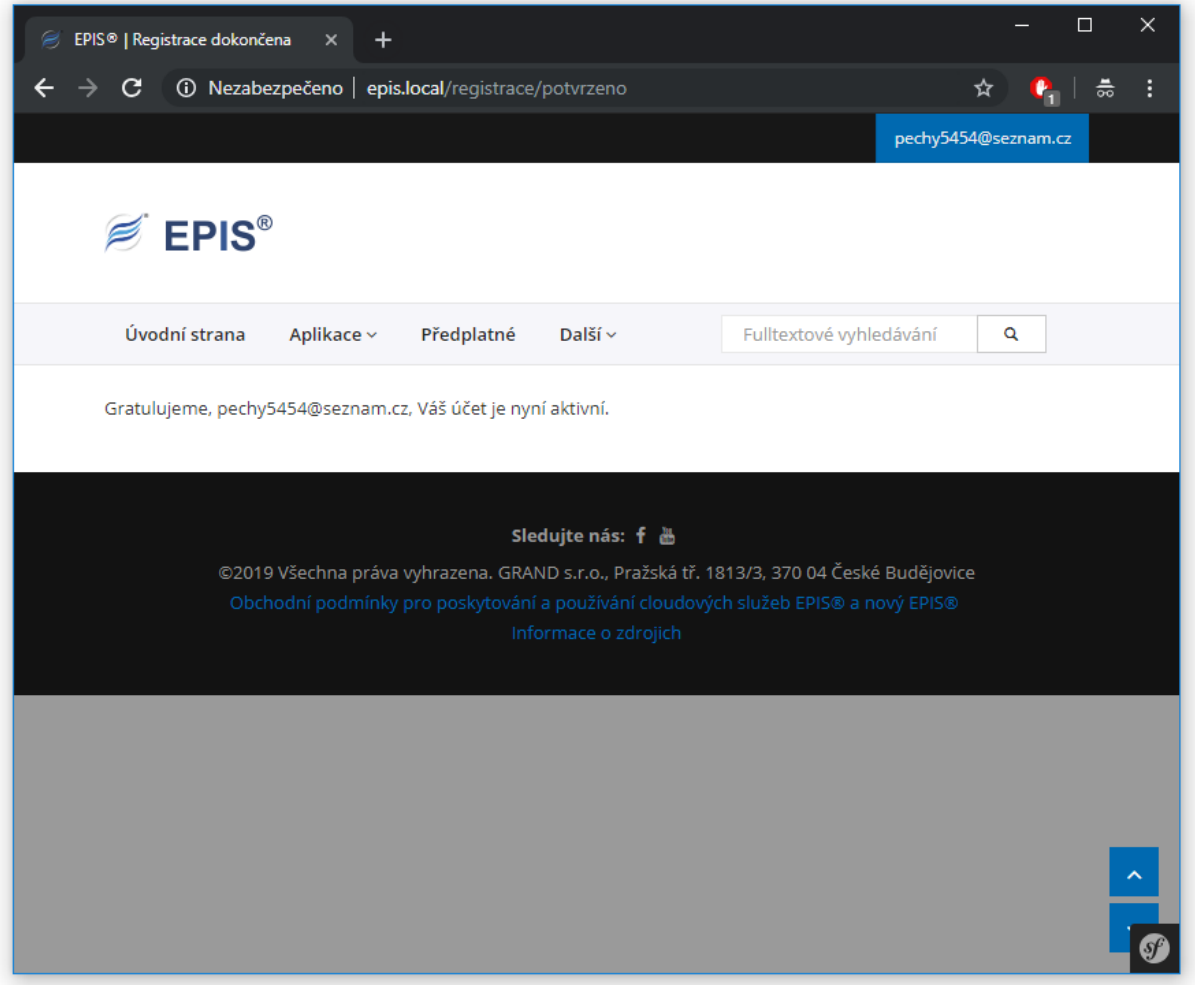
(Zdroj: https://novyepis.cz)

Příloha 21 – Obrazovka aplikace oznamující o úspěšné registraci



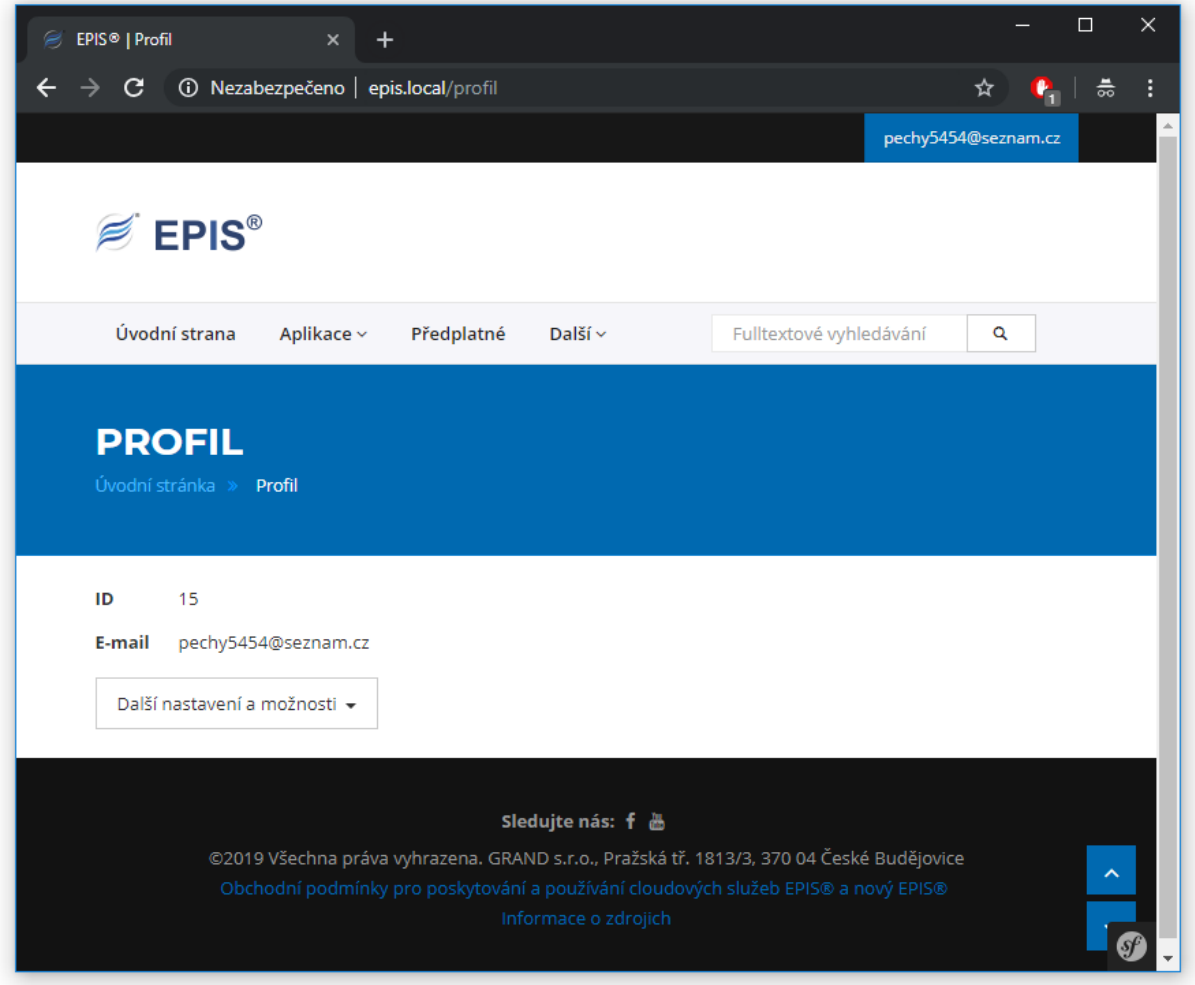
(Zdroj: https://novyepis.cz)

Příloha 22 – Obrazovka aplikace oznamující o aktivaci účtu



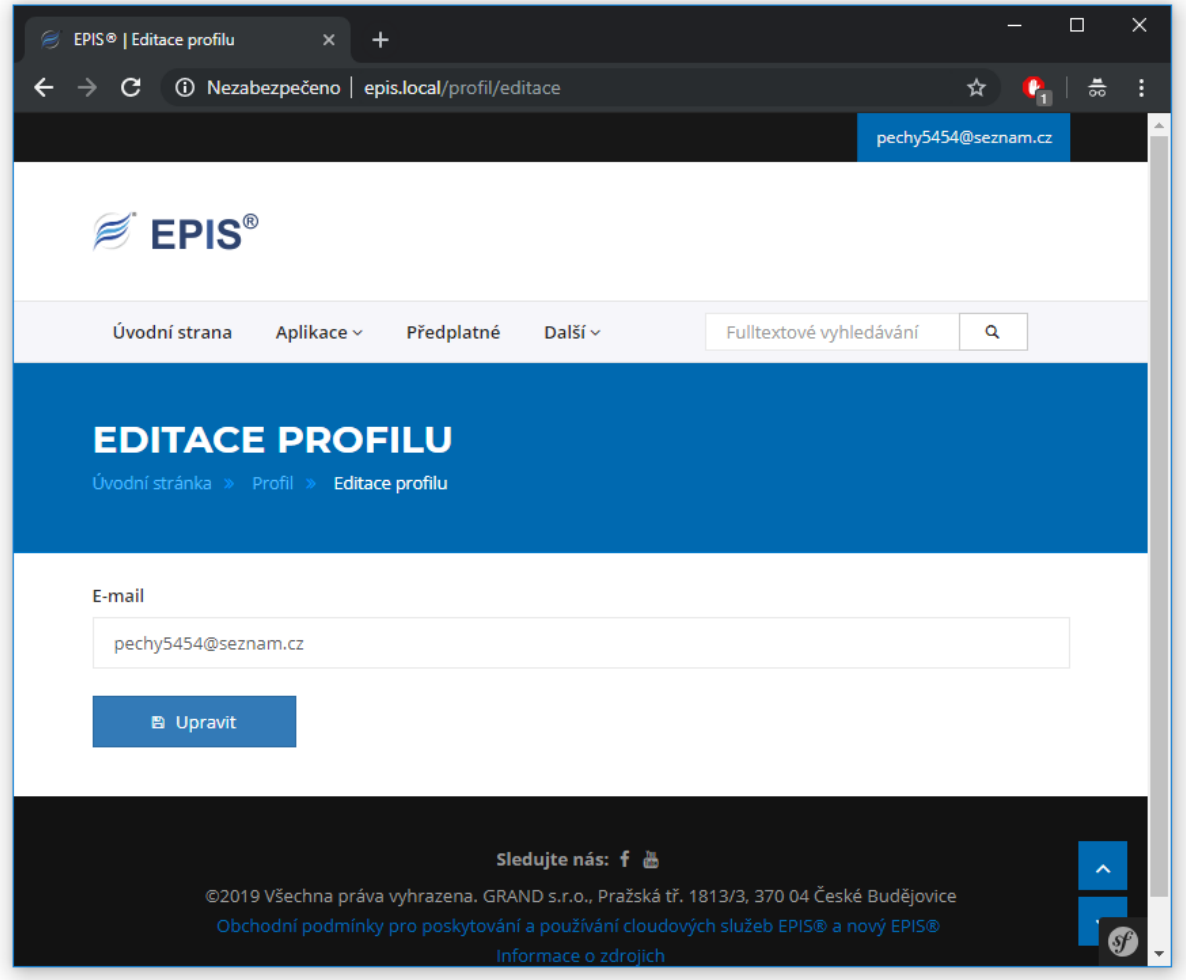
(Zdroj: https://novyepis.cz)

Příloha 23 – Obrazovka aplikace s profilem uživatele



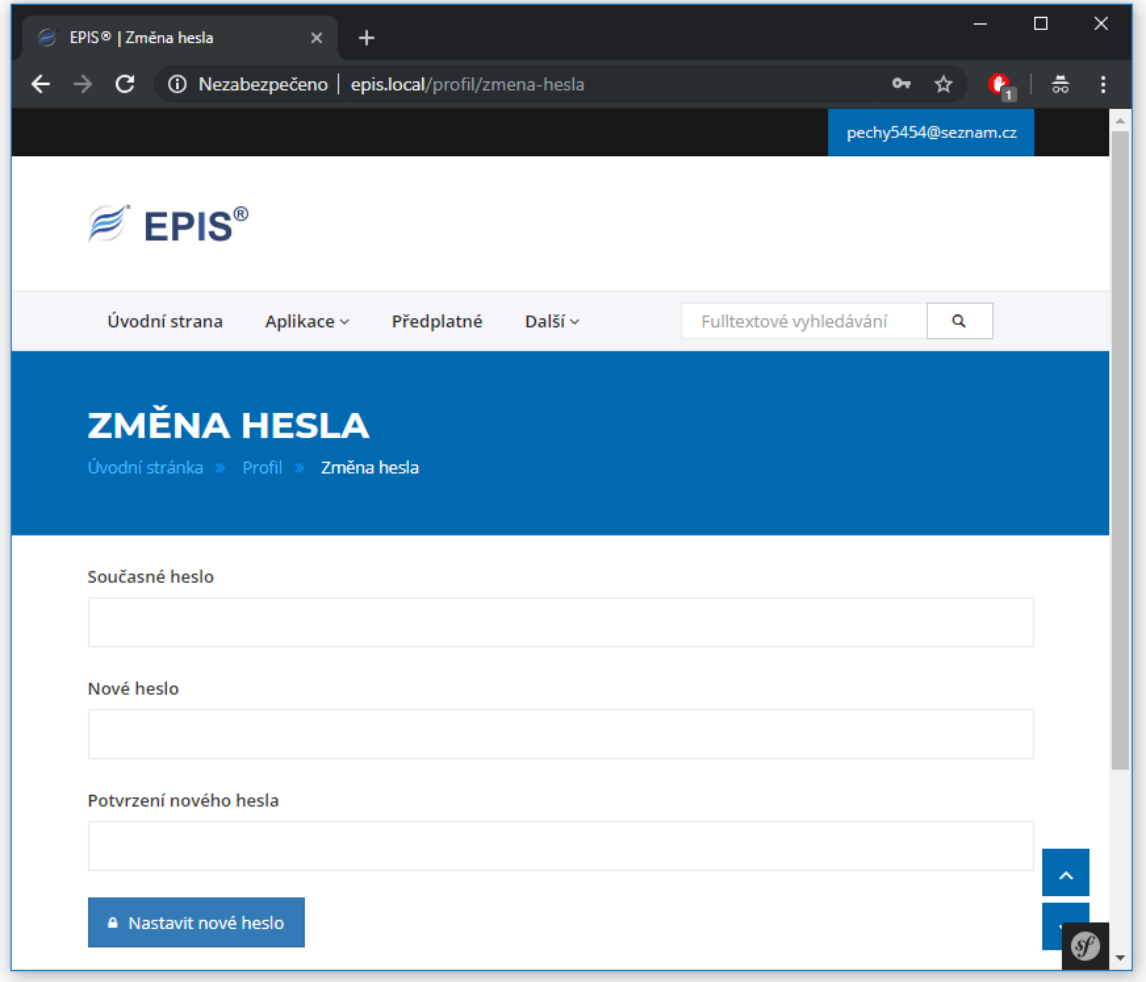
(Zdroj: https://novyepis.cz)

Příloha 24 – Obrazovka aplikace s editačním formulářem profilu uživatele



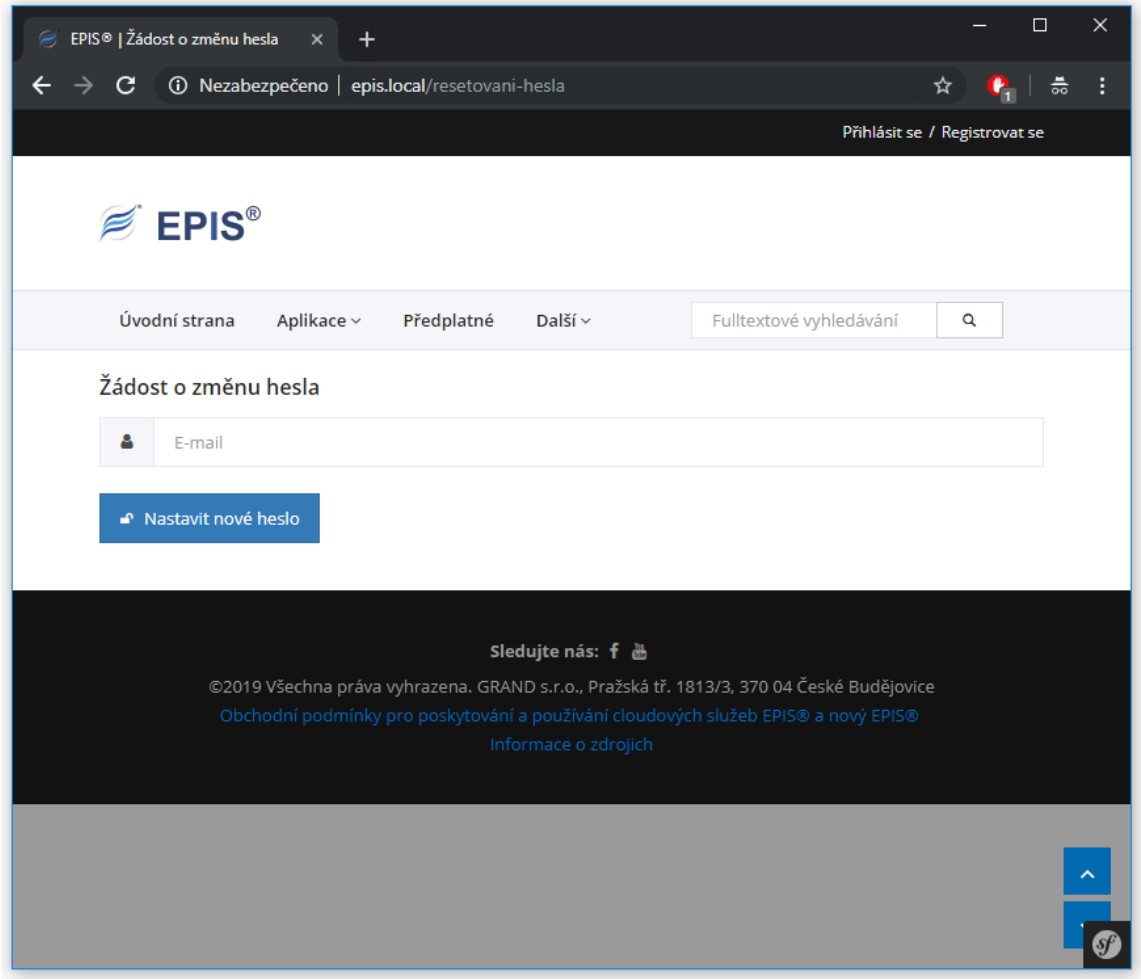
(Zdroj: https://novyepis.cz)

Příloha 25 – Obrazovka aplikace s formulářem umožňujícím změnu hesla



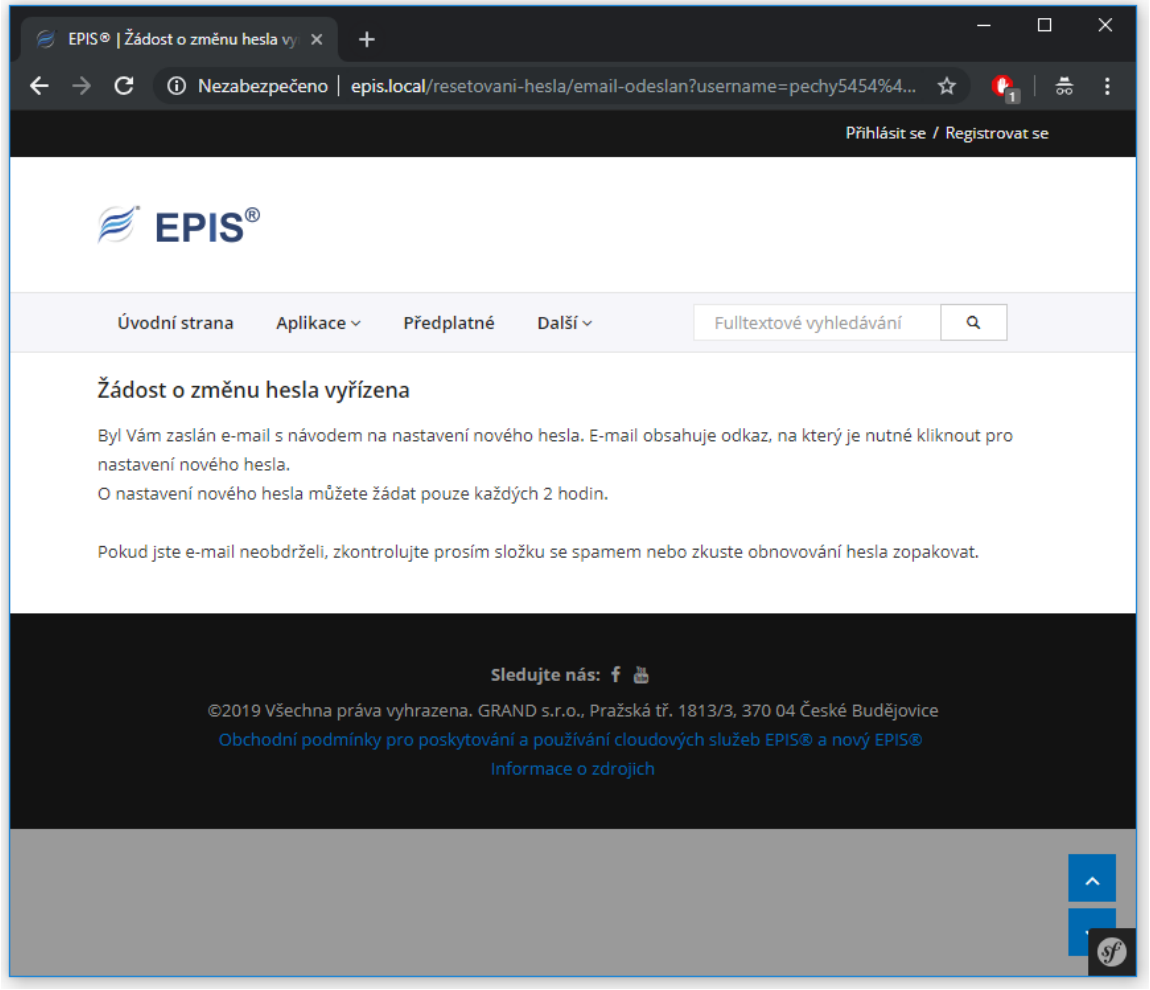
(Zdroj: https://novyepis.cz)

Příloha 26 – Obrazovka aplikace se žádostí o změnu zapomenutého hesla



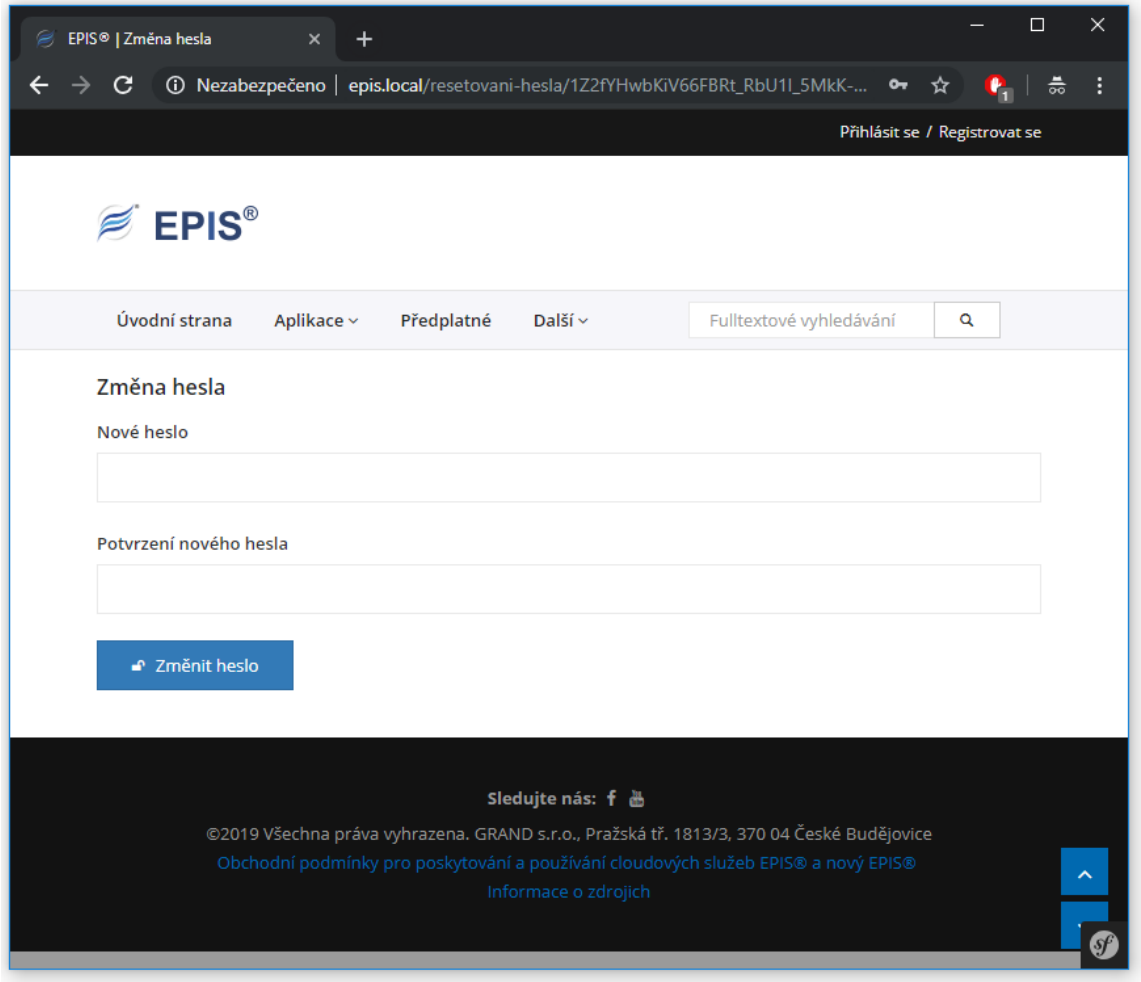
(Zdroj: https://novyepis.cz)

Příloha 27 – Obrazovka aplikace oznamující vyřízení žádosti o změnu zapomenutého hesla



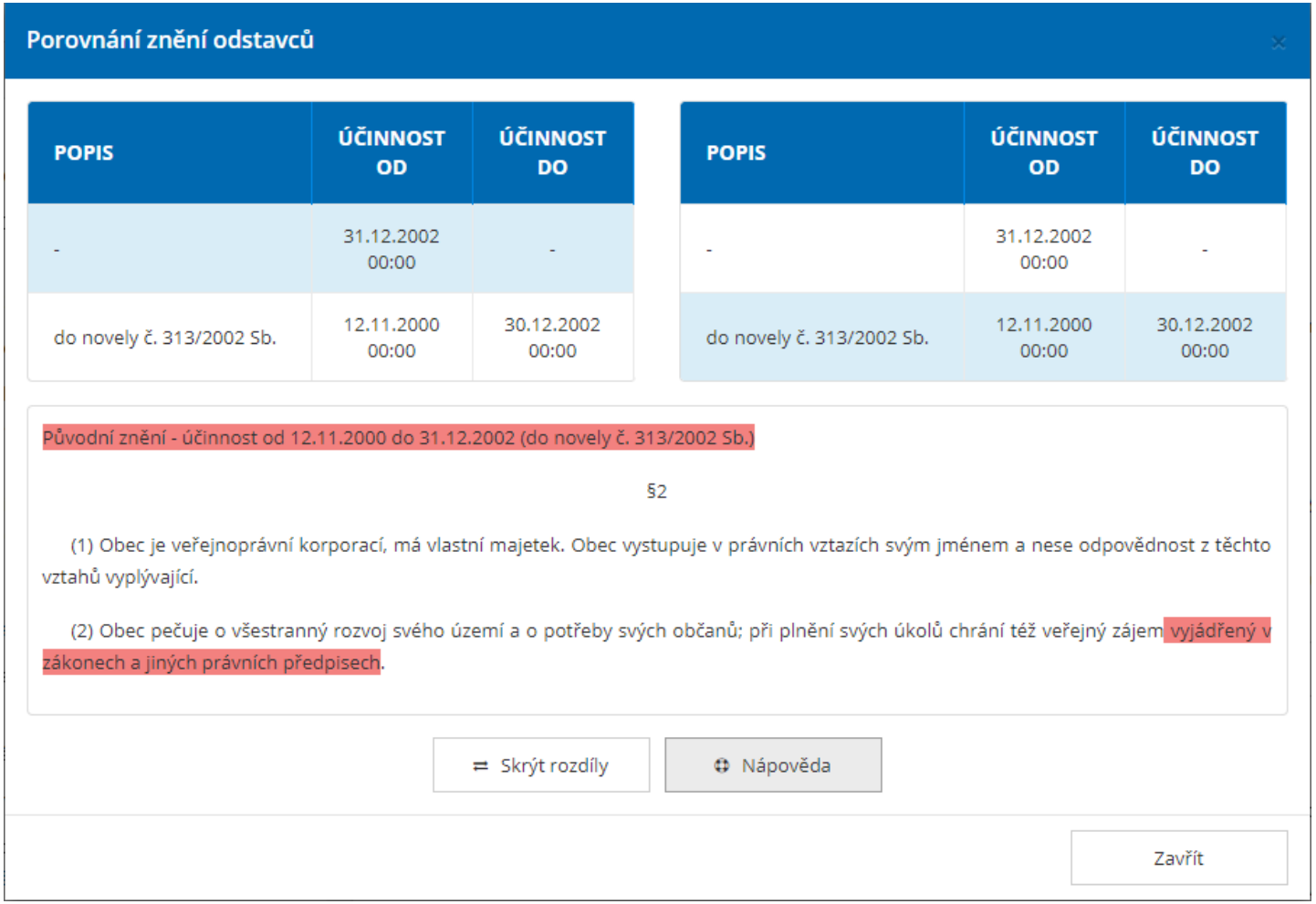
(Zdroj: https://novyepis.cz)

Příloha 28 – Obrazovka aplikace s formulářem umožňujícím změnu zapomenutého hesla



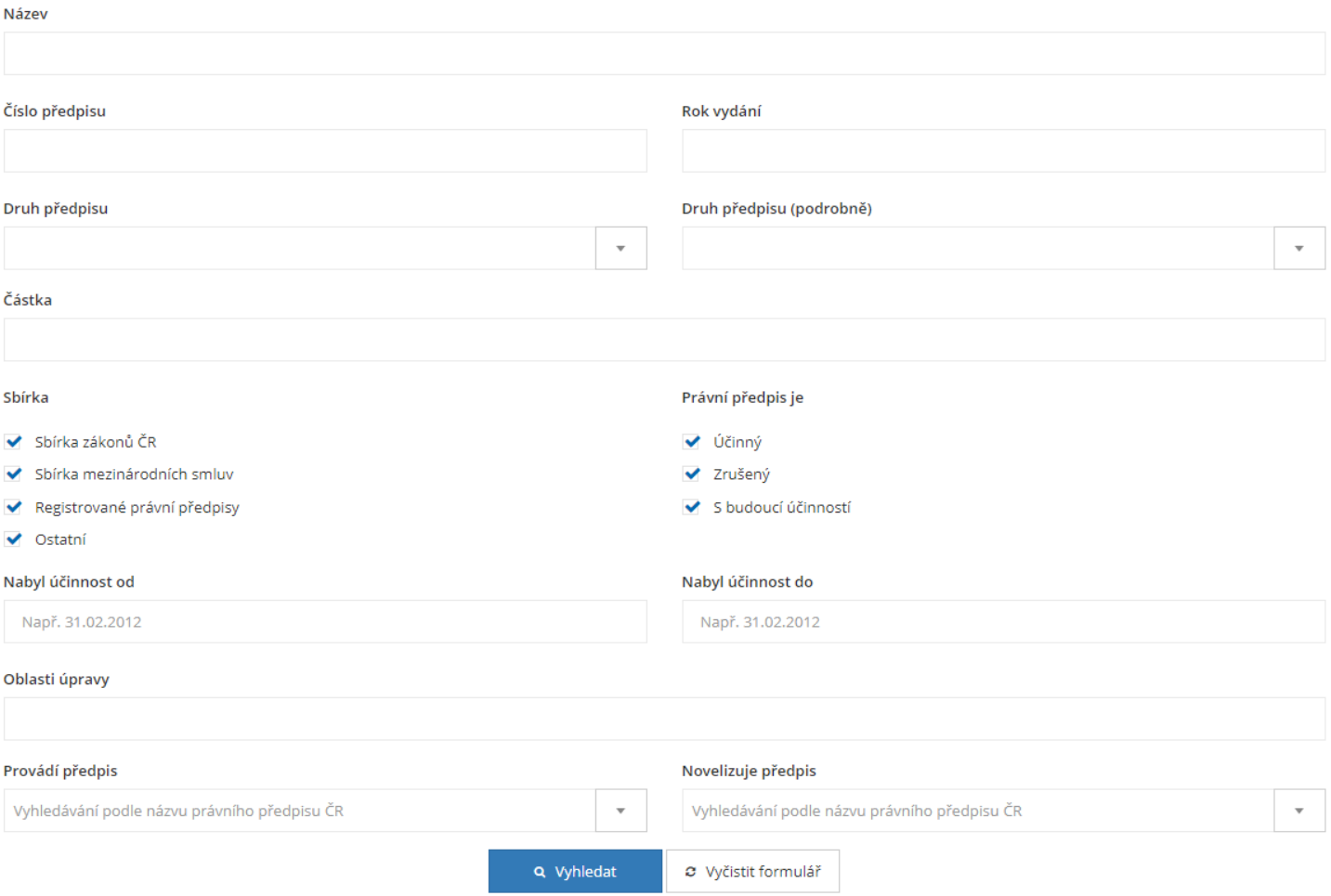
(Zdroj: https://novyepis.cz)

Příloha 29 – Porovnání znění jednotlivých verzí paragrafu právního předpisu



(Zdroj: https://novyepis.cz)

Příloha 30 – Vyhledávací formulář pro právní předpisy ČR



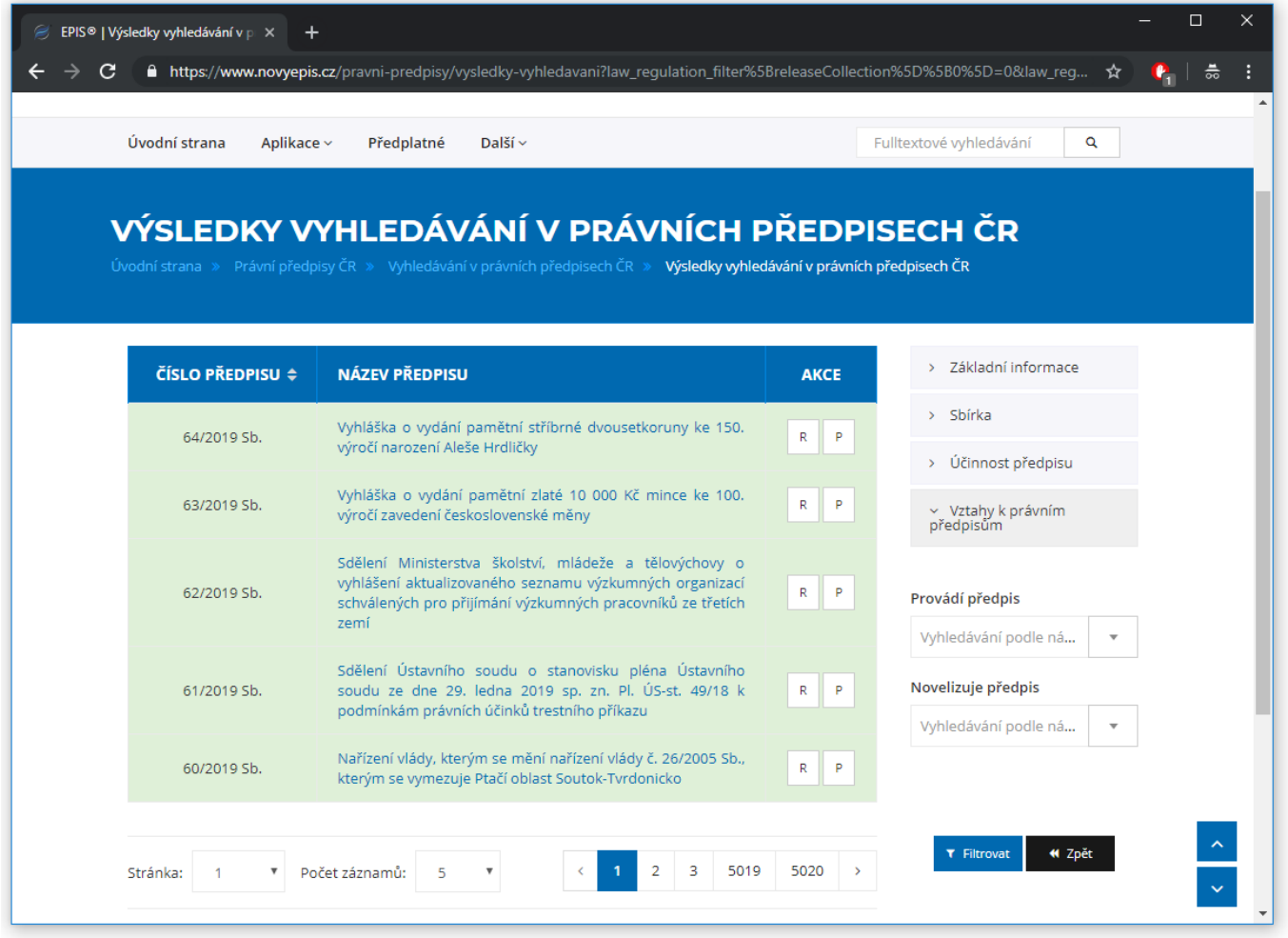
(Zdroj: https://novyepis.cz)

Příloha 31 - Metoda buildForm třídy LawRegulationFilterType

```php
public function buildForm(FormBuilderInterface $builder, array $options)
{
    $builder->add( child: "number", type: NumberFilterType::class, array(
        "scale" => 0,
        "label" => "Číslo předpisu"
    ));
    $builder->add( child: "year", type: NumberFilterType::class, array(
        "scale" => 0,
        "label" => "Rok vydání"
    ));
    $builder->add( child: "lawRegulationType", type: EntityFilterType::class, [
        "label"        => "Druh předpisu",
        "required"     => false,
        "multiple"     => false,
        "class"        => LawRegulationType::class,
        "choices"      => $this->entityManager->getRepository( className: LawRegulationType::class)->findAll(),
        "choice_label" => "name"
    ]);
    $builder->add( child: "lawRegulationKind", type: EntityFilterType::class, [
        "label"        => "Druh předpisu (podrobně)",
        "required"     => false,
        "multiple"     => false,
        "class"        => LawRegulationKind::class,
        "choices"      => $this->entityManager->getRepository( className: LawRegulationKind::class)->findAll(),
        "choice_label" => "name"
    ]);
    $builder->add( child: "name", type: TextFilterType::class, array(
        "label" => "Název"
    ));
    $builder->add( child: "figure", type: NumberFilterType::class, array(
        "label" => "Částka",
        "scale" => 0
    ));
    $builder->add( child: "releaseCollection", type: ChoiceFilterType::class, array(
        "label"         => "Sbírka",
        "multiple"      => true,
        "expanded"      => true,
        "choice_loader" => new CallbackChoiceLoader(function () {
            return array_flip([0 => "Sbírka zákonů ČR", 1 => "Sbírka mezinárodních smluv",
                               2 => "Registrované právní předpisy", 3 => "Ostatní"]);
        }),
        "data"          => array(0, 1, 2, 3)
    ));
    $builder->add( child: "effectiveFrom", type: DateRangeFilterType::class, array(
        "left_date_options"  => array(
            "label"  => "Nabyl účinnost od",
            "html5"  => false,
            "widget" => "single_text",
            "format" => "dd.MM.yyyy"
        ),
        "right_date_options" => array(
            "label"  => "Nabyl účinnost do",
            "html5"  => false,
            "widget" => "single_text",
            "format" => "dd.MM.yyyy"),
        "required"           => false
    ));
    $builder->add( child: "lawRegulationSections", type: EntityFilterType::class, [
        "label"        => "Oblasti úpravy",
        "required"     => true,
        "multiple"     => false,
        "class"        => LawRegulationSection::class,
        "choices"      => $this->entityManager->getRepository( className: LawRegulationSection::class)->findAll(),
        "choice_label" => "name"
    ]);
    $builder->add( child: "implementsLawRegulations", type: EntityFilterType::class, [
        "label"        => "Provádí předpis",
        "required"     => true,
        "multiple"     => false,
        "class"        => LawRegulation::class,
        "choices"      => $this->entityManager->getRepository( className: LawRegulation::class)->findAll(),
        "choice_label" => "name"
    ]);
    $builder->add( child: "amendsLawRegulations", type: EntityFilterType::class, [
        "label"        => "Novelizuje předpis",
        "required"     => true,
        "multiple"     => false,
        "class"        => LawRegulation::class,
        "choices"      => $this->entityManager->getRepository( className: LawRegulation::class)->findAll(),
        "choice_label" => "name"
    ]);
}
```

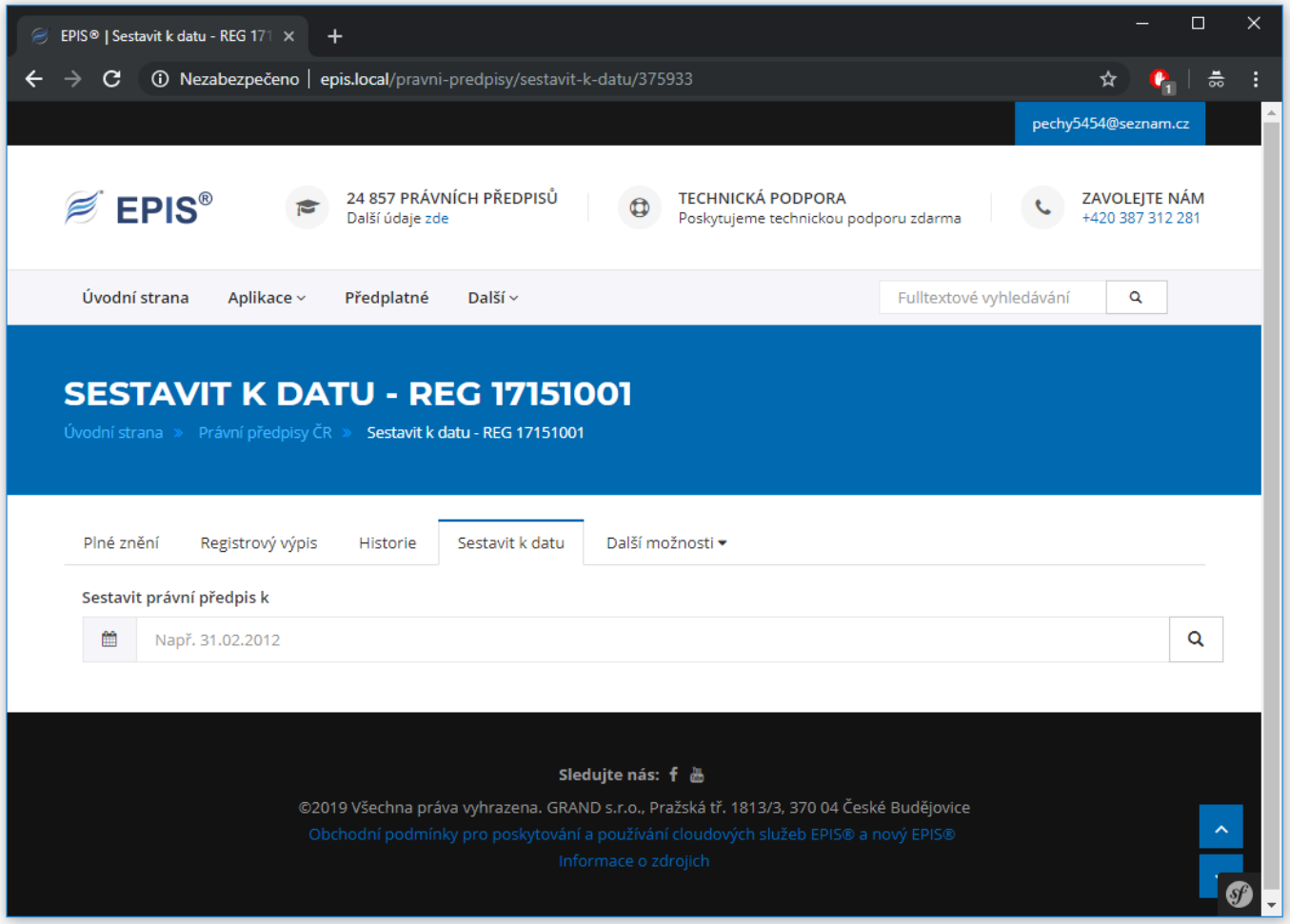
(Zdroj: Autor)

Příloha 32 – Stránkované výsledky vyhledávaní v právních předpisech



(Zdroj: https://novyepis.cz)

Příloha 33 – Obrazovka umožňující sestavení právního předpisu ke zvolenému datu



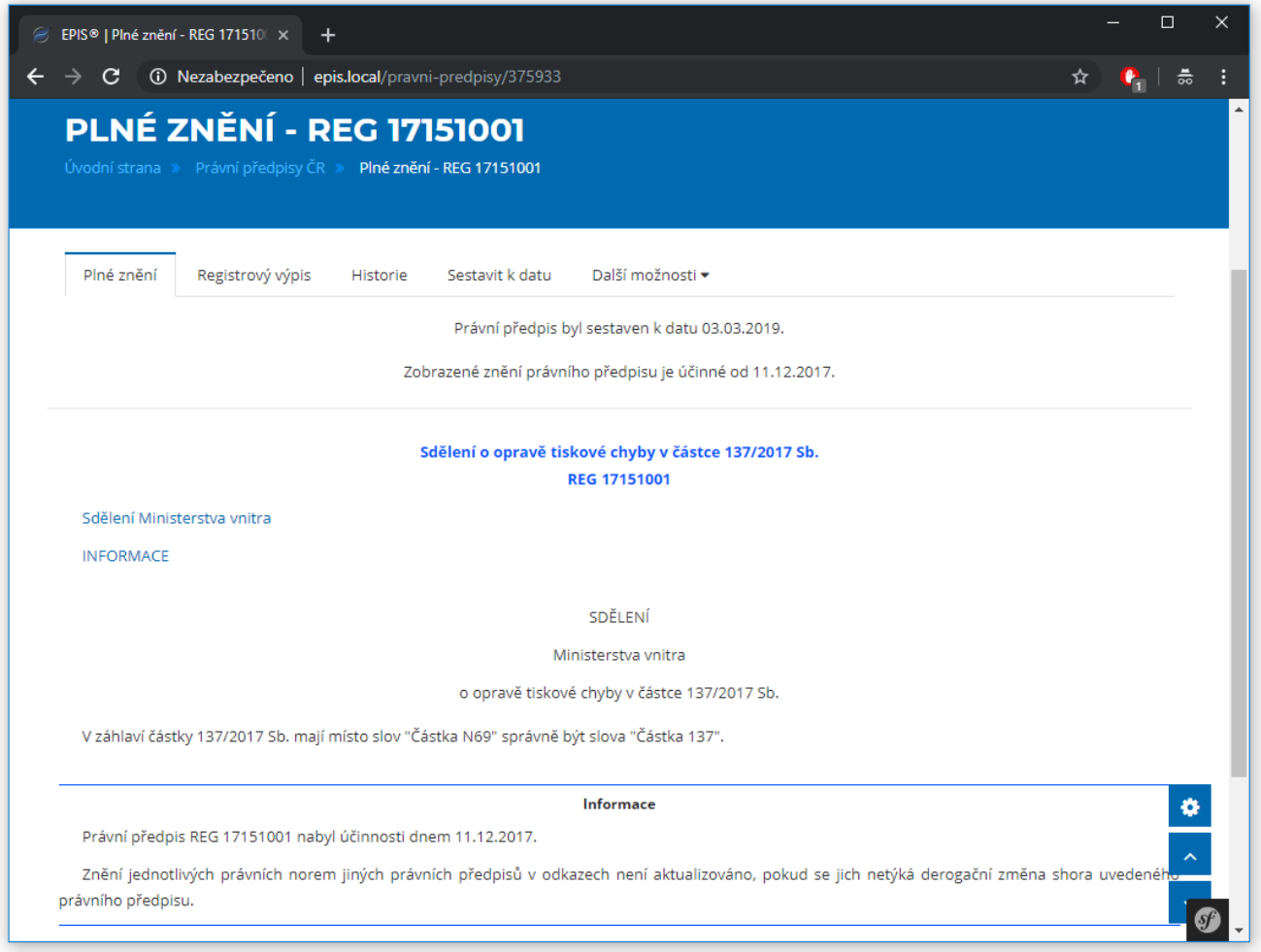
(Zdroj: https://novyepis.cz)

Příloha 34 – Akce řadiče zobrazující právní předpis ČR

```php
/**
 * @Route("/pravni-predpisy/{id}", name="epis_law_regulation_article", requirements={"id": "\d+"})
 * @Method("GET")
 */
public function articleAction(LawRegulation $lawRegulation, Request $request)
{
    if ($lawRegulation->getArticle() == null)
    {
        throw $this->createNotFoundException("Právní předpis nemá plné znění.");
    }
    $passedCompilationDate = \DateTime::createFromFormat( format: "d.m.Y", $request->get( key: "compilation-date"));
    $passedCompilationDate = $passedCompilationDate === false ? null : $passedCompilationDate;
    $usedCompilationDate = $passedCompilationDate == null ? (new \DateTime())->setTime( hour: 0,  minute: 0) : $passedCompilationDate;
    if ($passedCompilationDate == null && $lawRegulation->getEffectiveUntil() != null && $usedCompilationDate > $lawRegulation->getEffectiveUntil())
    {
        $usedCompilationDate = $lawRegulation->getEffectiveUntil();
    }
    else if ($passedCompilationDate == null && $lawRegulation->getEffectiveFrom() != null
        && $usedCompilationDate < $lawRegulation->getEffectiveFrom())
    {
        $usedCompilationDate = $lawRegulation->getEffectiveFrom();
    }
    if (($passedCompilationDate != null && $lawRegulation->getEffectiveUntil() != null
            && $passedCompilationDate > $lawRegulation->getEffectiveUntil())
        || ($passedCompilationDate != null && $lawRegulation->getEffectiveFrom() != null && $passedCompilationDate < $lawRegulation->getEffectiveFrom()))
    {
        throw $this->createNotFoundException("Právní předpis k požadovanému datu neexistuje.");
    }
    return $this->render( view: "@Epis/Register/LawRegulation/article", array(
        "law_regulation"   => $lawRegulation,
        "paragraphs"       => $this->getDoctrine()->getRepository( persistentObject: Paragraph::class)->getLawRegulationParagraphs($lawRegulation, $usedCompilationDate),
        "compilation_date" => $usedCompilationDate,
        "censor"           => !$this->get("security.authorization_checker")->isGranted( attributes: "VIEW", $lawRegulation)
    ));
}
```

(Zdroj: Autor)

Příloha 35 – Obrazovka zobrazující právní předpis ČR



(Zdroj: https://novyepis.cz)

Příloha 36 – Definice třídy GlobalRegisterVoter

```
class GlobalRegisterVoter extends Voter
{
    const VIEW = "view";

    /** @var AccessDecisionManagerInterface */
    private $decisionManager;

    /** @var ObjectManager */
    private $mainObjectManager;

    /** @var RequestStack */
    private $request;

    /**
     * GlobalRegisterVoter constructor.
     * @param AccessDecisionManagerInterface $decisionManager
     * @param ObjectManager                  $mainObjectManager
     * @param RequestStack                   $requestStack
     */
    public function __construct(AccessDecisionManagerInterface $decisionManager, ObjectManager $mainObjectManager, RequestStack $requestStack)
    {
        $this->decisionManager = $decisionManager;
        $this->mainObjectManager = $mainObjectManager;
        $this->request = $requestStack->getMasterRequest();
    }

    protected function supports($attribute, $subject){...}

    protected function voteOnAttribute($attribute, $subject, TokenInterface $token){...}

    /** @param GlobalRegisterItem $globalRegisterItem ...*/
    private function isGlobalRegisterItemForFree($globalRegisterItem){...}

    /** @param GlobalRegisterItem $globalRegisterItem ...*/
    private function isGlobalRegisterItemAccessBought($globalRegisterItem, $user){...}

    /** @return string ...*/
    private function getUrlPrefix(){...}
}
```

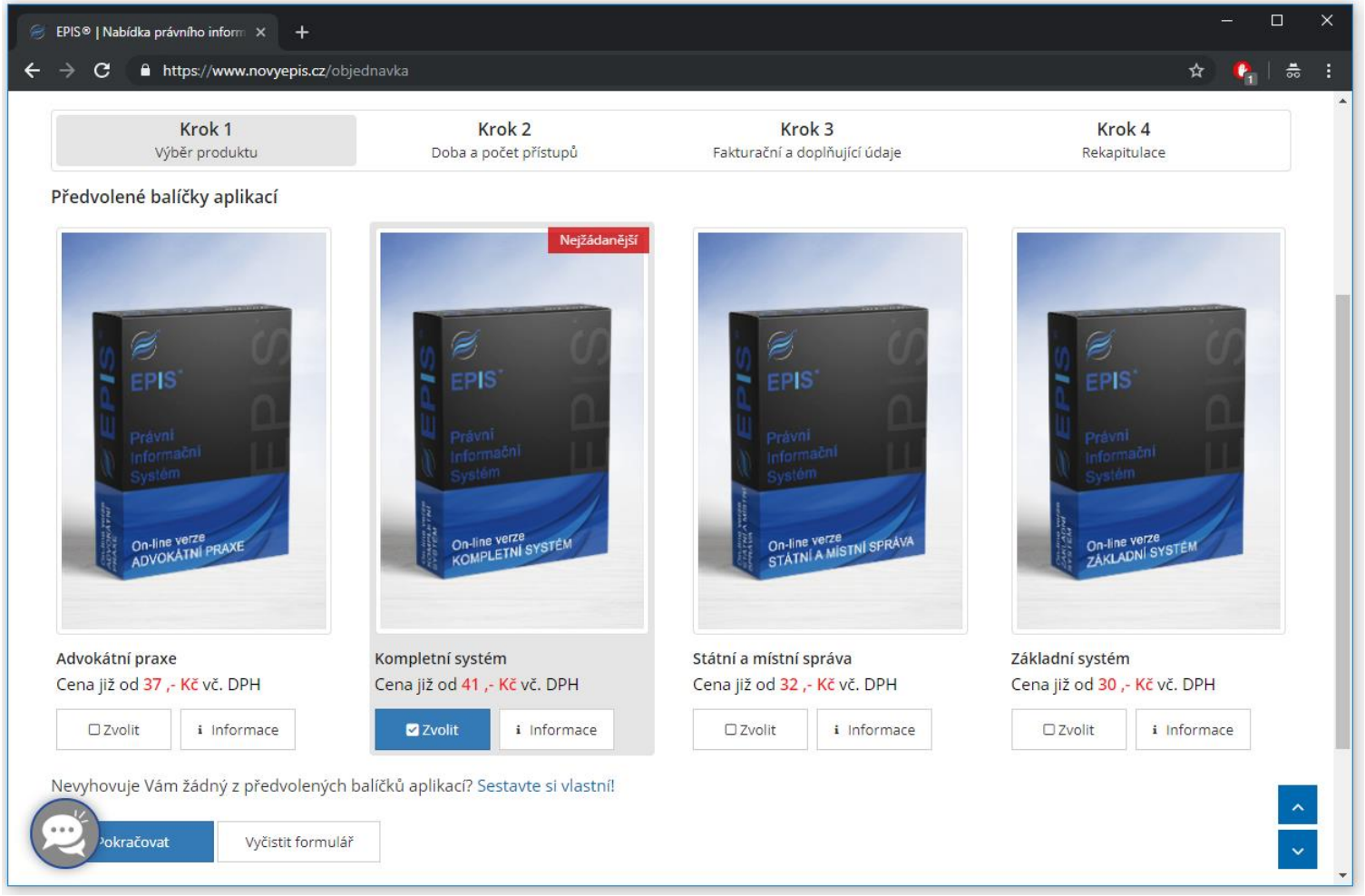
(Zdroj: Autor)

Příloha 37 – Metoda isGlobalRegisterItemAccessBought třídy GlobalRegisterVoter

```php
/**
 * @param GlobalRegisterItem $globalRegisterItem
 * @param User               $user
 * @return bool
 */
private function isGlobalRegisterItemAccessBought($globalRegisterItem, $user)
{
    $qb = $this->mainObjectManager->getRepository( className: User::class)->createQueryBuilder( alias: "user");
    $qb->select( select: "COUNT(license.id)")
        ->join( join: "user.activations", alias: "activations")
        ->join( join: "activations.license", alias: "license")
        ->leftJoin( join: "license.modules", alias: "modules")
        ->join( join: "modules.urlPrefixes", alias: "url_prefixes")
        ->andWhere("user = :user")->setParameter( key: "user", $user)
        ->andWhere("user.enabled = :user_enabled")->setParameter( key: "user_enabled", value: true)
        ->andWhere("activations.enabled = :activations_enabled")->setParameter( key: "activations_enabled", value: true)
        ->andWhere("license.enabled = :license_enabled")->setParameter( key: "license_enabled", value: true)
        ->andWhere("license.validFrom <= :license_valid_from")->setParameter( key: "license_valid_from", (new \DateTime())->setTime( hour: 0, minute: 0), type: Type::DATE)
        ->andWhere("license.validTo >= :license_valid_to")->setParameter( key: "license_valid_to", (new \DateTime())->setTime( hour: 0, minute: 0), type: Type::DATE)
        ->andWhere("modules.enabled = :modules_enabled AND url_prefixes.value = :url_prefix")
        ->setParameter( key: "modules_enabled", value: true)->setParameter( key: "url_prefix", $this->getUrlPrefix()));

    return ArrayHelper::firstValue($qb->getQuery()->getResult()[0]) > 0;
}
```

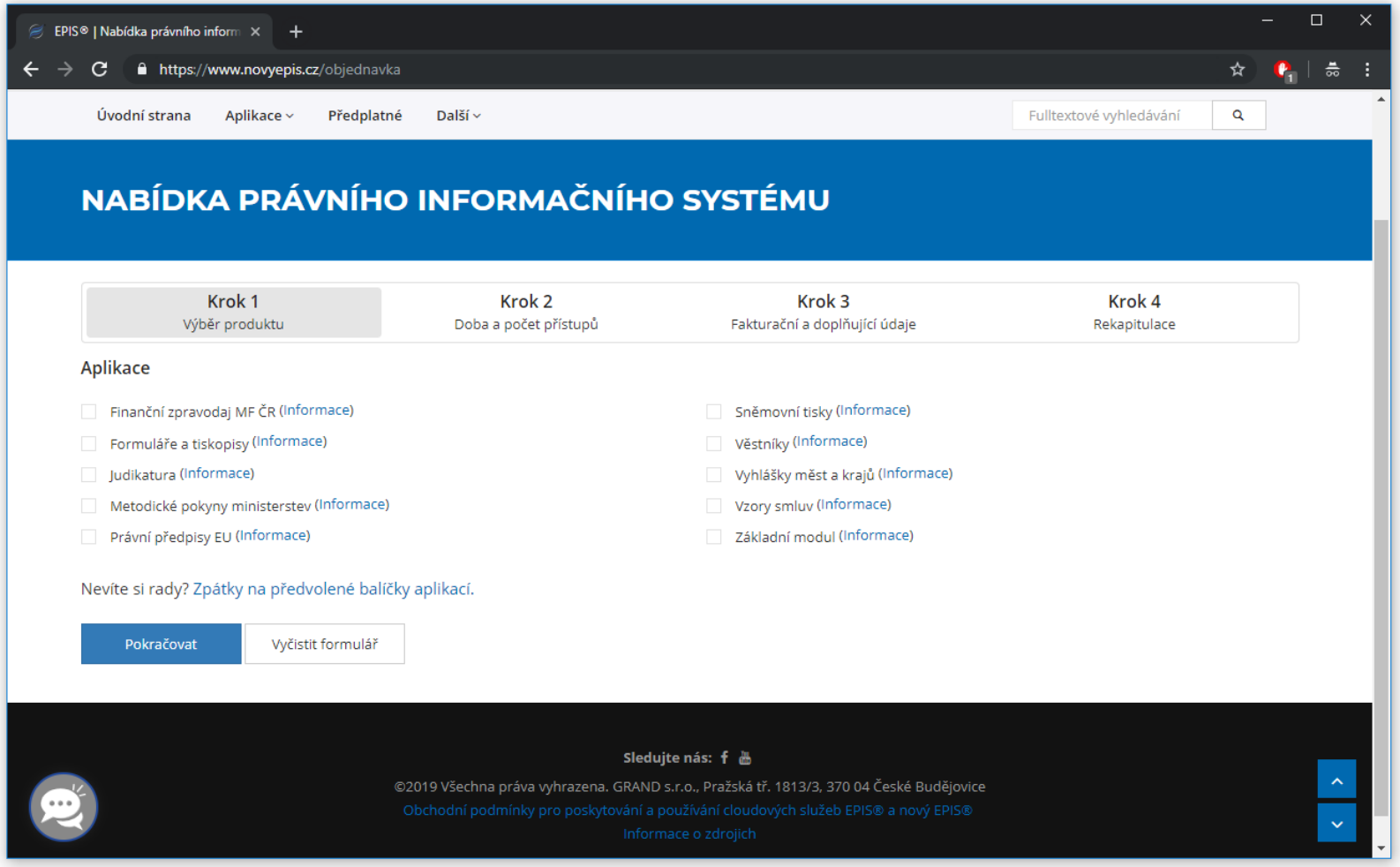
(Zdroj: Autor)

Příloha 38 – První krok objednávkového formuláře (výběr předvolené skupiny balíčků)



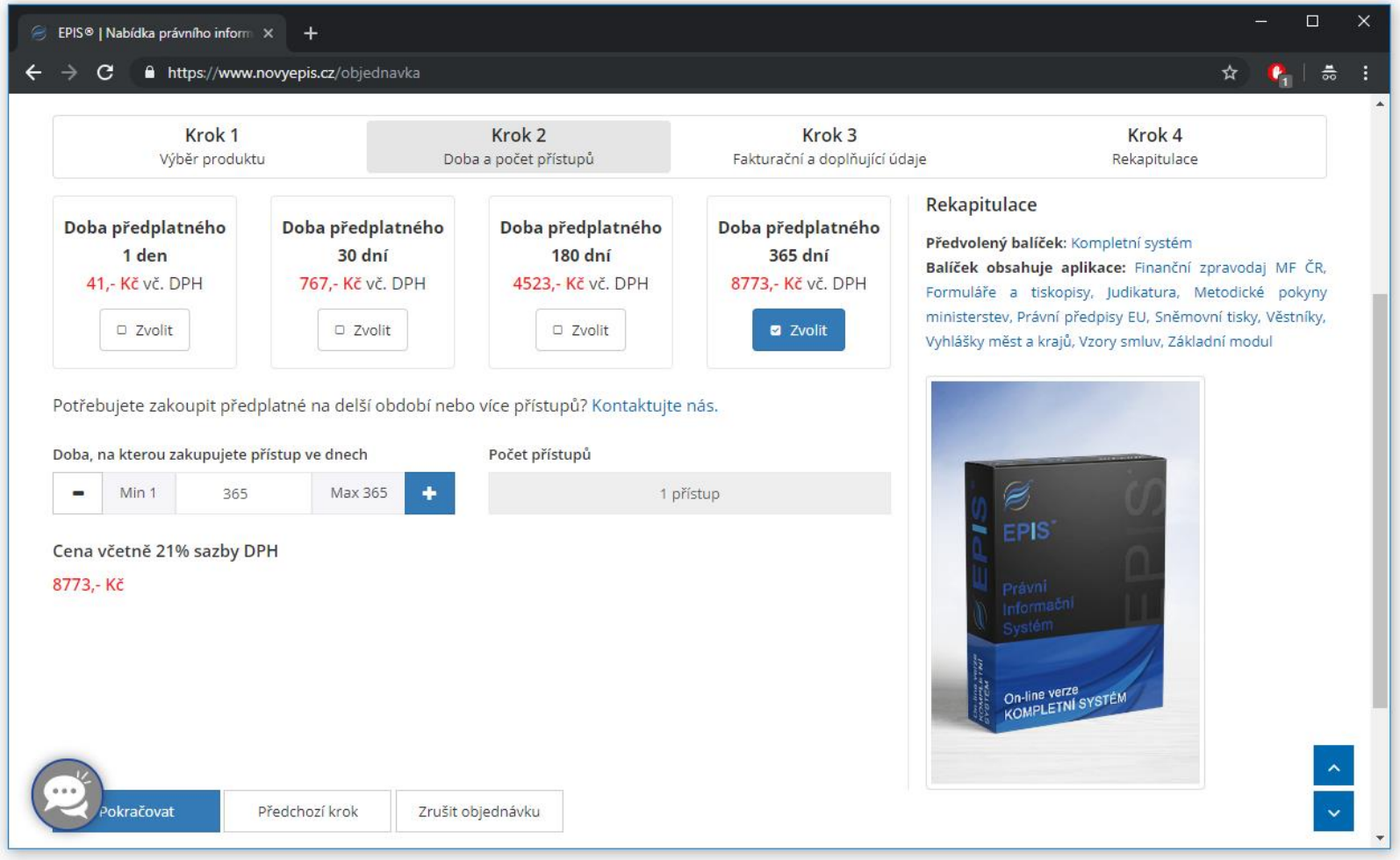
(Zdroj: https://novyepis.cz)

Příloha 39 - První krok objednávkového formuláře (výběr modulů)



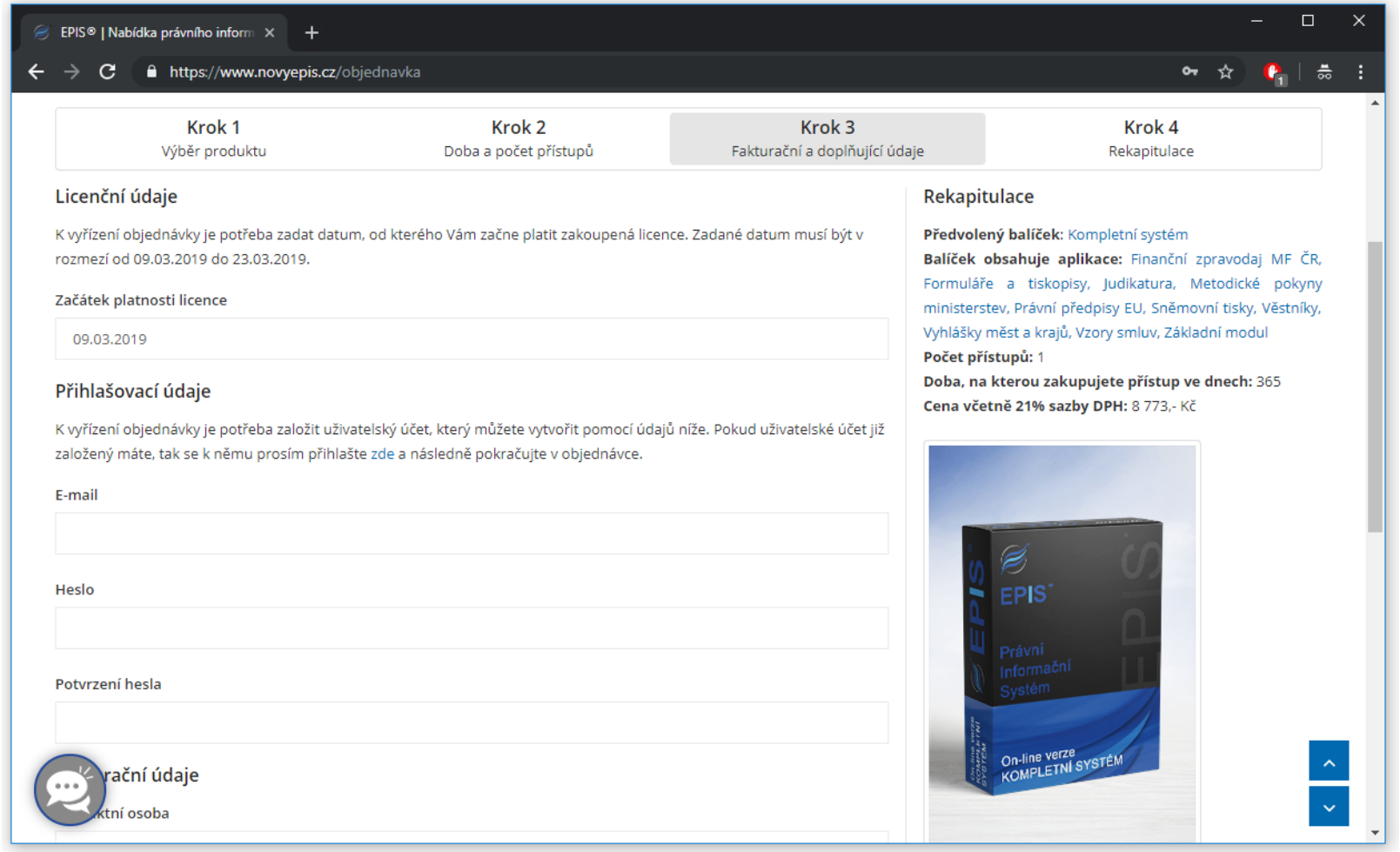
(Zdroj: https://novyepis.cz)

Příloha 40 – Druhý krok objednávkového formuláře



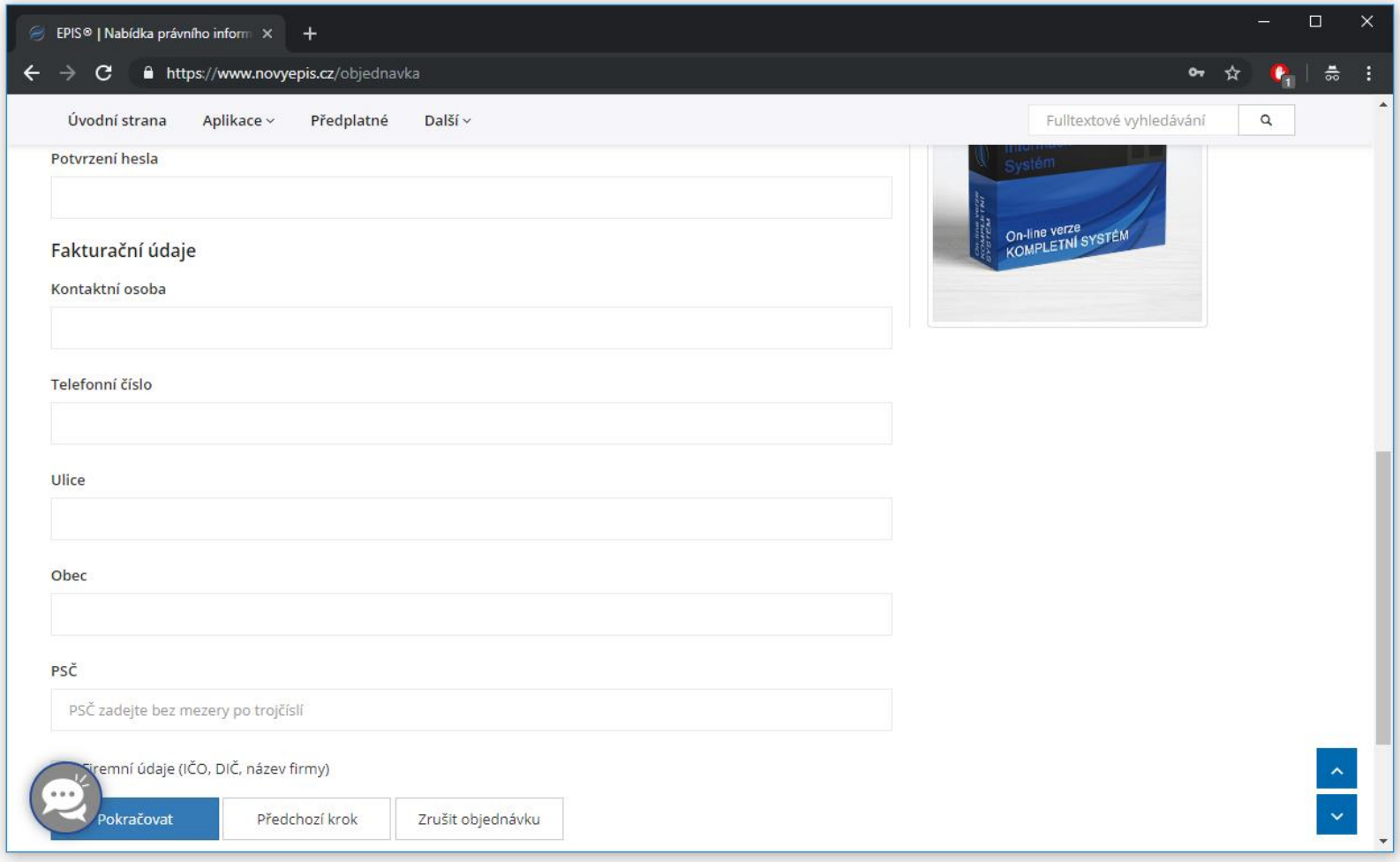
(Zdroj: https://novyepis.cz)

Příloha 41 – Třetí krok objednávkového formuláře (první část)



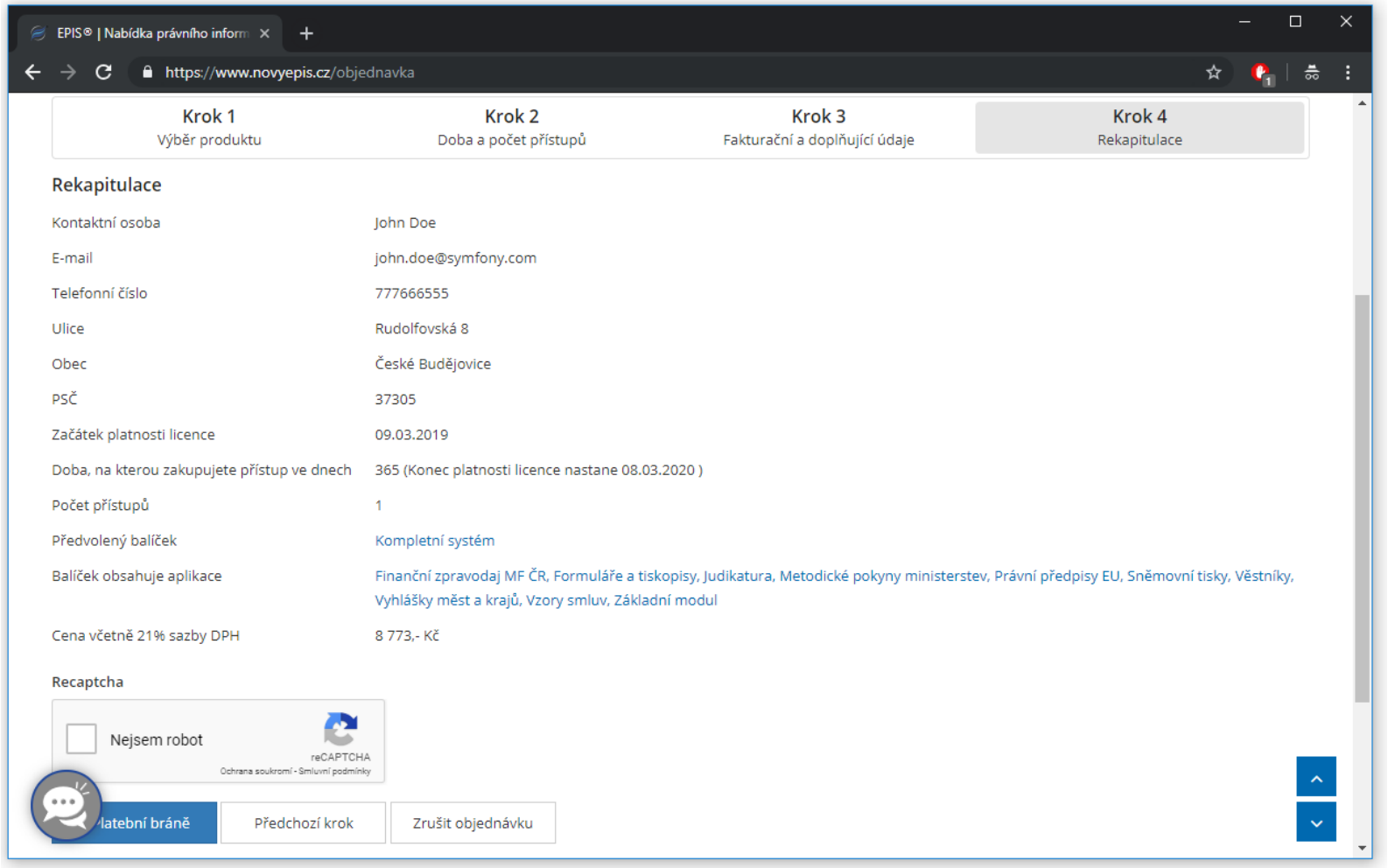
(Zdroj: https://novyepis.cz)

Příloha 42 – Třetí krok objednávkového formuláře (druhá část)



(Zdroj: https://novyepis.cz)

Příloha 43 – Čtvrtý krok objednávkového formuláře



(Zdroj: https://novyepis.cz)

Příloha 44 – Definice objednávkového formuláře

```php
namespace EpisBundle\Form\Main;


use ...

class OrderType extends AbstractType
{
    /** @var User */
    private $user;

    /** @var string */
    private $environment;

    /** @var PurchaseManager */
    private $purchaseManager;

    /**
     * @param TokenStorageInterface $tokenStorage
     * @param string                $environment
     */
    public function __construct(TokenStorageInterface $tokenStorage, string $environment, PurchaseManager $purchaseManager)
    {
        $this->user = $tokenStorage->getToken()->getUser() instanceof User ? $tokenStorage->getToken()->getUser() : null;
        $this->environment = $environment;
        $this->purchaseManager = $purchaseManager;
    }

    public function buildForm(FormBuilderInterface $builder, array $options){...}

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            "data_class" => Order::class
        ]);
    }
}
```

(Zdroj: Autor)

Příloha 45 – Akce řadiče OrderController vypočítávající cenu objednávky podle zadaných parametrů

```php
/**
 * @Route("/objednavka/cena", name="epis_purchase_modules_order_price", options={"expose": "true"}, defaults={"_format": "json"})
 * @Method("GET")
 */
public function modulesOrderPriceAction(Request $request, ObjectManager $mainObjectManager, PurchaseManager $purchaseManager)
{
    $period = $request->get( key: "period");
    $numberOfActivations = $request->get( key: "numberOfActivations");
    $ownModulesSelection = $request->get( key: "ownModulesSelection");
    $modulesGroup = $mainObjectManager->getRepository( className: ModulesGroup::class)->findOneBy(["id" => $request->get( key: "modulesGroup")]);
    $modules = $mainObjectManager->getRepository( className: Module::class)->findBy(["id" => $request->get( key: "modules")]);
    $order = (new Order())->setOwnModulesSelection($ownModulesSelection)->setModulesGroup($modulesGroup);
    foreach ($modules as $module)
    {
        $order->addModule($module);
    }
    $order->setNumberOfActivations($numberOfActivations)->setPeriod($period);
    $purchaseManager->correctModulesOrder($order);
    if (!$order->isModulesOrderValid())
    {
        return new JsonResponse(["status" => false, "message" => "You have passed wrong arguments."]);
    }

    return new JsonResponse([
        "status" => true,
        "price"  => $purchaseManager->calculatePrice($order)
    ]);
}
```

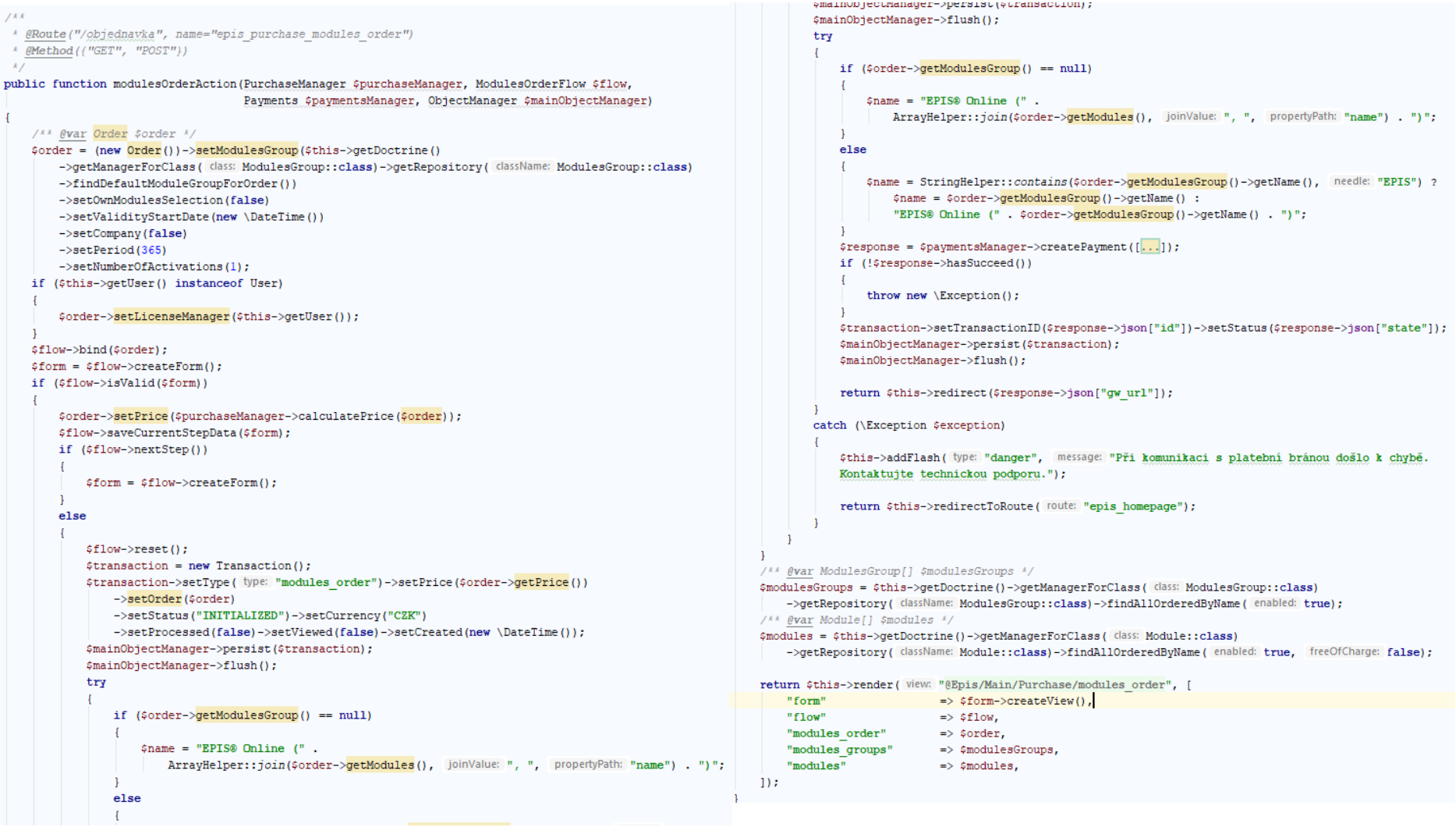
(Zdroj: Autor)

Příloha 46 – Akce řadiče OrderController zobrazující stav transakce

```php
/**
 * @Route("/objednavka/{id}", name="epis_purchase_oder_show", requirements={"id": "\d+"})
 * @Breadcrumb("Úvodní strana", routeName="epis_homepage")
 * @Method("GET")
 */
public function orderShowAction(Transaction $transaction, Trail $breadcrumbsTrail, ObjectManager $mainObjectManager, Payments $paymentsManager)
{
    $response = $paymentsManager->getStatus($transaction->getTransactionID());
    if (!$response->hasSucceed())
    {
        throw $this->createNotFoundException();
    }
    $transaction->setStatus($response->json["state"]);
    $breadcrumbsTrail->add( breadcrumb_or_title: "Objednávka {$transaction->getId()}");
    $response = $this->render( view: "@Epis/Main/Purchase/order_show", [
        "transaction" => $transaction,
    ]);
    $transaction->setViewed( viewed: true);
    $mainObjectManager->persist($transaction);
    $mainObjectManager->flush();

    return $response;
}
```

(Zdroj: Autor)

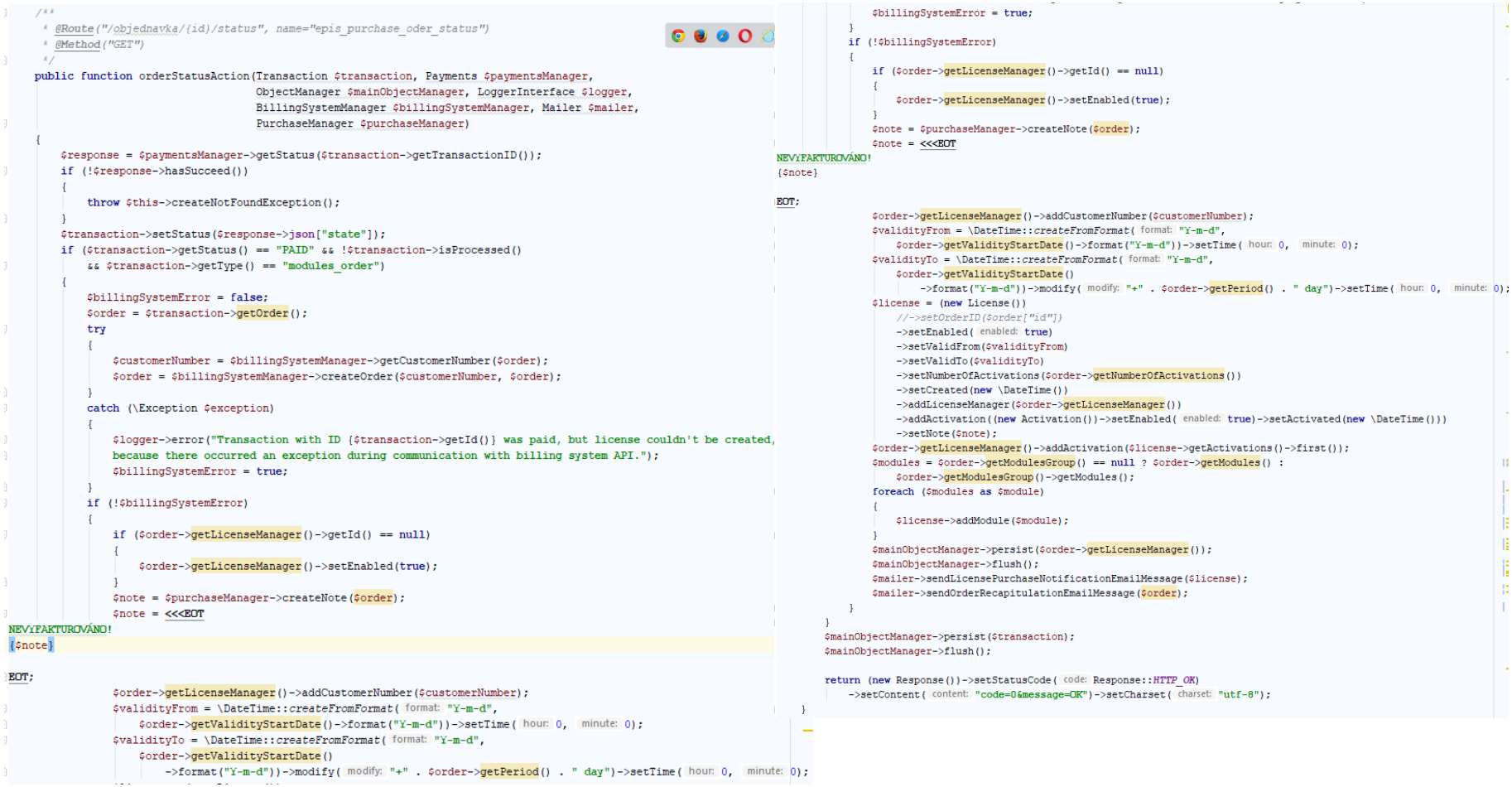
Příloha 47 – Akce řadiče OrderController obsluhující objednávkový formulář



```php
/**
 * @Route("/objednavka", name="epis_purchase_modules_order")
 * @Method({"GET", "POST"})
 */
public function modulesOrderAction(PurchaseManager $purchaseManager, ModulesOrderFlow $flow,
                                   Payments $paymentsManager, ObjectManager $mainObjectManager)
{
    /** @var Order $order */
    $order = (new Order())->setModulesGroup($this->getDoctrine()
        ->getManagerForClass( class: ModulesGroup::class)->getRepository( className: ModulesGroup::class)
        ->findDefaultModuleGroupForOrder())
        ->setOwnModulesSelection(false)
        ->setValidityStartDate(new \DateTime())
        ->setCompany(false)
        ->setPeriod(365)
        ->setNumberOfActivations(1);
    if ($this->getUser() instanceof User)
    {
        $order->setLicenseManager($this->getUser());
    }
    $flow->bind($order);
    $form = $flow->createForm();
    if ($flow->isValid($form))
    {
        $order->setPrice($purchaseManager->calculatePrice($order));
        $flow->saveCurrentStepData($form);
        if ($flow->nextStep())
        {
            $form = $flow->createForm();
        }
        else
        {
            $flow->reset();
            $transaction = new Transaction();
            $transaction->setType( type: "modules_order")->setPrice($order->getPrice())
                ->setOrder($order)
                ->setStatus("INITIALIZED")->setCurrency("CZK")
                ->setProcessed(false)->setViewed(false)->setCreated(new \DateTime());
            $mainObjectManager->persist($transaction);
            $mainObjectManager->flush();
            try
            {
                if ($order->getModulesGroup() == null)
                {
                    $name = "EPIS® Online (" .
                        ArrayHelper::join($order->getModules(), joinValue: ", ", propertyPath: "name") . ")";
                }
                else
                {
                    $name = StringHelper::contains($order->getModulesGroup()->getName(), needle: "EPIS") ?
                        $name = $order->getModulesGroup()->getName() :
                        "EPIS® Online (" . $order->getModulesGroup()->getName() . ")";
                }
                $response = $paymentsManager->createPayment([...]);
                if (!$response->hasSucceed())
                {
                    throw new \Exception();
                }
                $transaction->setTransactionID($response->json["id"])->setStatus($response->json["state"]);
                $mainObjectManager->persist($transaction);
                $mainObjectManager->flush();

                return $this->redirect($response->json["gw_url"]);
            }
            catch (\Exception $exception)
            {
                $this->addFlash( type: "danger", message: "Při komunikaci s platební bránou došlo k chybě.
                Kontaktujte technickou podporu.");

                return $this->redirectToRoute( route: "epis_homepage");
            }
        }
    }
    /** @var ModulesGroup[] $modulesGroups */
    $modulesGroups = $this->getDoctrine()->getManagerForClass( class: ModulesGroup::class)
        ->getRepository( className: ModulesGroup::class)->findAllOrderedByName( enabled: true);
    /** @var Module[] $modules */
    $modules = $this->getDoctrine()->getManagerForClass( class: Module::class)
        ->getRepository( className: Module::class)->findAllOrderedByName( enabled: true, freeOfCharge: false);

    return $this->render( view: "@Epis/Main/Purchase/modules_order", [
        "form"              => $form->createView(),
        "flow"              => $flow,
        "modules_order"     => $order,
        "modules_groups"    => $modulesGroups,
        "modules"           => $modules,
    ]);
}
```

(Zdroj: Autor)

Příloha 48 – Akce řadiče OrderController komunikující na pozadí s platební branou GoPay



```
    /**
     * @Route("/objednavka/{id}/status", name="epis_purchase_oder_status")
     * @Method("GET")
     */
    public function orderStatusAction(Transaction $transaction, Payments $paymentsManager,
                                      ObjectManager $mainObjectManager, LoggerInterface $logger,
                                      BillingSystemManager $billingSystemManager, Mailer $mailer,
                                      PurchaseManager $purchaseManager)
    {
        $response = $paymentsManager->getStatus($transaction->getTransactionID());
        if (!$response->hasSucceed())
        {
            throw $this->createNotFoundException();
        }
        $transaction->setStatus($response->json["state"]);
        if ($transaction->getStatus() == "PAID" && !$transaction->isProcessed()
            && $transaction->getType() == "modules_order")
        {
            $billingSystemError = false;
            $order = $transaction->getOrder();
            try
            {
                $customerNumber = $billingSystemManager->getCustomerNumber($order);
                $order = $billingSystemManager->createOrder($customerNumber, $order);
            }
            catch (\Exception $exception)
            {
                $logger->error("Transaction with ID {$transaction->getId()} was paid, but license couldn't be created,
                because there occurred an exception during communication with billing system API.");
                $billingSystemError = true;
            }
            if (!$billingSystemError)
            {
                if ($order->getLicenseManager()->getId() == null)
                {
                    $order->getLicenseManager()->setEnabled(true);
                }
                $note = $purchaseManager->createNote($order);
                $note = <<<EOT
NEVYFAKTUROVÁNO!
{$note}

EOT;
                $order->getLicenseManager()->addCustomerNumber($customerNumber);
                $validityFrom = \DateTime::createFromFormat( format: "Y-m-d",
                    $order->getValidityStartDate()->format("Y-m-d"))->setTime( hour: 0,  minute: 0);
                $validityTo = \DateTime::createFromFormat( format: "Y-m-d",
                    $order->getValidityStartDate()
                        ->format("Y-m-d"))->modify( modify: "+" . $order->getPeriod() . " day")->setTime( hour: 0,  minute: 0);
                $billingSystemError = true;
            }
            if (!$billingSystemError)
            {
                if ($order->getLicenseManager()->getId() == null)
                {
                    $order->getLicenseManager()->setEnabled(true);
                }
                $note = $purchaseManager->createNote($order);
                $note = <<<EOT
NEVYFAKTUROVÁNO!
{$note}

EOT;
                $order->getLicenseManager()->addCustomerNumber($customerNumber);
                $validityFrom = \DateTime::createFromFormat( format: "Y-m-d",
                    $order->getValidityStartDate()->format("Y-m-d"))->setTime( hour: 0,  minute: 0);
                $validityTo = \DateTime::createFromFormat( format: "Y-m-d",
                    $order->getValidityStartDate()
                        ->format("Y-m-d"))->modify( modify: "+" . $order->getPeriod() . " day")->setTime( hour: 0,  minute: 0);
                $license = (new License())
                    //->setOrderID($order["id"])
                    ->setEnabled( enabled: true)
                    ->setValidFrom($validityFrom)
                    ->setValidTo($validityTo)
                    ->setNumberOfActivations($order->getNumberOfActivations())
                    ->setCreated(new \DateTime())
                    ->addLicenseManager($order->getLicenseManager())
                    ->addActivation((new Activation())->setEnabled( enabled: true)->setActivated(new \DateTime()))
                    ->setNote($note);
                $order->getLicenseManager()->addActivation($license->getActivations()->first());
                $modules = $order->getModulesGroup() == null ? $order->getModules() :
                    $order->getModulesGroup()->getModules();
                foreach ($modules as $module)
                {
                    $license->addModule($module);
                }
                $mainObjectManager->persist($order->getLicenseManager());
                $mainObjectManager->flush();
                $mailer->sendLicensePurchaseNotificationEmailMessage($license);
                $mailer->sendOrderRecapitulationEmailMessage($order);
            }
        }
        $mainObjectManager->persist($transaction);
        $mainObjectManager->flush();

        return (new Response())->setStatusCode( code: Response::HTTP_OK)
            ->setContent( content: "code=0&message=OK")->setCharset( charset: "utf-8");
    }
```

(Zdroj: Autor)