Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Knowledge Management Center

Rozšíření informačního systému jazykové školy Lingua Nostra

Bc. Štěpán Kylberger

Supervisor: Ing. Pavel Náplava, Ph.D.
May 2021

Acknowledgements

Rád bych poděkoval svému vedoucímu práce Ing. Pavlu Náplavovi, Ph.D. za vedení bakalářské práce a za užitečné připomínky a rady k mé práci. V neposlední řadě bych rád poděkoval rodině a přátelům za jejich podporu při psaní práce. Dále bych chtěl poděkovat Renatě Skoupé za bezproblémovou spolupráci, Ondřeji Kašparovi za trpělivost a Lucille Baressi, že si na mě udělala čas.

Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Abstract

This work focuses on the problematic of language schools. Specifically it takes up the language school Lingua Nostra, documents it's organisation and functioning and identifies possible expansions of the current information system. Subsequently it implements a solution and evaluates its impact.

Keywords: Language School, Information System

Supervisor: Ing. Pavel Náplava, Ph.D.
ČVUT FEL,
Technická 2,
166 27 Praha 6 - Dejvice

Abstrakt

Tato práce se soustředí na problematiku jazykových škol. Konkrétně se zabývá jazykovou školou Lingua Nostra, dokumentuje její organizaci a fungování a identifikuje možná rozšíření stávajícího informačního systému. Následně implementuje řešení a hodnotí jeho dopad.

Klíčová slova: Jazyková škola, informační systém

Překlad názvu: Expansion of Lingua Nostra Language School's Information System

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

V této práci se zabývám implementací rozšíření stávajícího informačního systému jazykové školy Lingua Nostra. Téma jazykových škol je mi blízké, protože s nimi mám dlouholetou osobní zkušenost, nejdříve jako klient a v současné době jako lektor. Jazyková škola je komerční instituce, která najímá lektory a za úplatu učí klienty nějaký jazyk, ať už cizí, nebo češtinu pro cizince. Může se jednat jak o výuku čistě zájmovou, tak i kvalifikační, která slouží jako příprava na složení některé jazykové zkoušky nebo k nabytí jazykových schopností potřebných v zaměstnání. Měl jsem možnost nahlédnout fungování této instituce ze dvou stran a vidět, jakým způsobem může být vnější aspekt, tedy tvář, kterou jazyková škola ukazuje klientům, podporována a zajišťována aspektem vnitřního fungování, v němž jsou obsaženy interakce lektorů s klienty a také s administrativními pracovníky. Mechanismy tohoto vnitřního fungování nebývají nikterak banální: je třeba zajistit správné a rychlé toky informací, kvalitní produkt a bezpečné, pečlivé zacházení s financemi.[6, p. 150]

Malá jazyková škola může tuto agendu zvládnout pomocí tradičních „papírových“ prostředků, je však zřejmé, že od určitého počtu klientů, a tedy i lektorů a administrativních pracovníků, začne být evidence pomocí těchto prostředků neúměrně neflexibilní a náročná. V takové situaci se nabízí podpořit fungování informačním systémem. Je nicméně třeba dodat, že koupě softwaru sama o sobě zřídka vyřeší problémy organizace, pokud se nezmění její fungování.

V tomto prostředí se běžně pohybuji, věnuji podstatnou část svého času kromě výuky i administrativním agendám, v některých případech pociťuji dopad jejich neefektivnosti a z vlastní zkušenosti vím, jak vážné důsledky může tato neefektivnost na jazykové školy mít: zvyšovat náklady, zpomalovat důležité procesy nebo vést až ke stavu úplné paralýzy, kdy další fungování zkrátka není možné.

Během studia SIT jsem absolvoval předměty zaměřené na informační systémy a procesní řízení a modelové situace, které jsme zde probírali, mnohdy odpovídaly právě takovýmto případům, a proto jsem se rozhodl tyto zatím teoreticky nabyté znalosti prakticky využít. Jako cíl této práce jsem tedy stanovil následující: nejprve najít jazykovou školu, jejíž provoz brzdí neúměrně náročné administrativní procesy a která zároveň buď nemá žádný informační systém, nebo jen takový, který poskytuje nedostatečnou podporu; a následně

pomocí rozšíření stávajícího nebo implementace nového informačního systému zmíněné komplikované procesy zjednodušit tak, aby pro danou jazykovou školu přestaly být problémem.

Nyní stručně uvedu strukturu práce. V kapitole Analýza dosavadního fungování popíši nejprve proces výběru konkrétní jazykové školy, s níž jsem spolupracoval, dále se budu věnovat problematice jazykových škol obecně a poté konkrétnímu případu vybrané školy. Nejdříve přiblížím její organizaci a business model a v závěru této kapitoly popíši stávající stav podpůrných systémů, které využívá. V další kapitole „Oblasti fungování stávajícího systému vhodné pro rozšíření podpory“ se budu věnovat diskusi funkcionalit, které se zdály být vhodnými kandidáty na rozšíření a výslednému výběru. Obsahem kapitoly „Vybraná rozšíření stávajícího systému“ pak bude bližší popis účelu a nástin fungování těchto funkcionalit. Kapitola „Popis řešení rozšiřujících funkcionalit“ se bude věnovat technické stránce implementace těchto funkcionalit, nejprve zvolené softwarové architektuře - konkrétně architektuře vrstevnaté - a následně jejím vrstvám, počínaje tou spodní, persistenční, přes vrstvu aplikační logiky až po vrstvu prezentační. Ve druhé polovině této kapitoly se budu věnovat bezpečnostním aspektům implementovaných funkcionalit a použitým technologiím a následně jejich testování. Závěrečná kapitola „Zhodnocení výsledků“ bude obsahovat zpětnou vazbu vedení jazykové školy, vyhodnocení projektové práce a následně některé návrhy možného budoucího směřování a lessons learned.

Kapitola 2

Analýza dosavadního fungování Lingua Nostra

Logickým prvním krokem práce byl výběr jazykové školy, s níž budu spolupracovat. Abych jí však mohl dodat produkt, který pro ni bude přínosný, bylo nutné nejprve zjistit, jaké jsou vlastně její potřeby. Tato otázka je složitější, než se na první pohled jeví, a proto bych se rád procesu toho, jak jsem na ni odpovídal a odpovědi samotné věnoval obsáhleji.

Jako nejjednodušší cesta se sice nabízí se prostě dotázat vedení: „Co byste potřebovali, aby váš podnik lépe fungoval?“ a dostat odpověď typu “moc bychom chtěli chatbota, který by za nás odpovídal na e-maily klientů,“ avšak reálně hrozí, že takový přístup by ke zdárnému výsledku nevedl. Klienti mají sklon vyjadřovat své potřeby pomocí řešení [7, p. 194], které sice může být dobré, ale rozhodně nemusí být nejlepší. Rozhodl jsem se proto postupovat důkladněji. Nejprve jsem popsal business model Lingua Nostra a její organizaci: klíčové procesy, které zde probíhají, a aktéry, jenž se jich účastní. Tato fáze probíhala formou konzultací s užším vedením a některými zaměstnanci. Následně jsem zmapoval podporu těchto procesů stávajícími informačními systémy. Tu jsem konzultoval s IT pracovníkem, avšak spíše povrchně. Těžiště tohoto mapování spočívalo ve studiu samotného stávajícího podpůrného systému, a to jak jako uživatel, tak čtením kódu.

2.1 Výběr konkrétní školy

Jazykovou školou, na niž se v této práci soustředím, je Lingua Nostra, pražská jazyková škola zaměřená na výuku italštiny a pořádání zážitkových akcí souvisejících s italskou kulturou. Nebyla to však má první volba. Jako první jsem oslovil jinou pražskou jazykovou školu, která fungovala čistě na bázi klasické papírové evidence, zdála se tedy být ideálním kandidátem na nějakou formu automatizace provozních agend, např. právě zavedení informačního systému. Avšak tato jazyková škola byla již v době, kdy jsem oslovil jejího zástupce, přetížena administrativními činnostmi do té míry, že nebyla schopna poskytnout součinnost v podobě pravidelných konzultací. K tomuto přetížení přispěla kromě jiného i nedávná pandemie. Druhá jazyková škola, již jsem oslovil, byla v opačné situaci. Informační systém již měla a tento byl intenzivně

využíván pro širokou škálu agend jak administrativními pracovníky, tak lektory i samotnými studenty. Nebyla zde tedy pro mne příležitost provést další automatizaci, nicméně se přinejmenším jednalo o konkrétní příklad jazykové školy, která do svého fungování informační systém úspěšně integrovala. Další institucí již jsem oslovil byla právě Lingua Nostra. Tato jazyková škola prošla historickým vývojem naznačeným výše - z počátku měla jen několik kurzů, a proto zcela postačovala tradiční evidence, která se však s rostoucím počtem studentů stala neudržitelnou. Vedení se tedy rozhodlo přejít na elektronickou evidenci a následně na jednoduchý, svépomocí zhotovený informační systém. Tento byl však vyvíjen spíše nahodile, se zřetelem pouze k aktuálním potřebám. Činnost Lingua Nostra se ale rozvíjí do stále nových oblastí a vyvstává potřeba podpořit nové netriviální agendy, které nebyly dříve předvídány. Vyhodnotil jsem tedy právě tuto instituci jako vhodné prostředí pro svou bakalářskou práci a navázal s ní kontakt, její vedení nabídku přijalo a poskytlo součinnost.

2.2 Jazykové školy obecně

Abychom pochopili, které otázky a proč bylo třeba si klást, bude vhodné popsat jazykovou školu jako instituci detailněji, než jak bylo nastíněno v Úvodu. Výuka může být směřována spíše zájmově nebo spíše kvalifikačně. O čistě binární rozdělení se nejedná, existují i jazykové školy, které provozují oba typy výuky, namátkou lze uvést například Chinese Point. Tato škola se soustředí na asijské jazyky a nabízí jak pomaturitní studium, tak třeba kurzy pro děti. Kritérium kvalifikační vs. zájmová výuka ovšem není jediné, jazykové školy také obvykle člení kurzy na skupinové a individuální, podle věku klientů, úrovně pokročilosti znalostí, zda výuku vede rodilý mluvčí či česky mluvící lektor nebo podle tematického zaměření.

Samotná výuka je pak v režii lektorů, kteří jsou zaměstnanci jazykové školy. Ne vždy se jedná o hlavní pracovní poměr, běžné jsou částečné úvazky nebo dohody o pracovní činnosti. Lektoři typicky bývají buď rodilí mluvčí žijící v dané lokalitě, nebo absolventi/studenti filologických oborů, nebo stážisté ze zahraničních univerzit (namnoze studenti pedagogických oborů). Z pohledu klientů je poptávka jak po rodilých mluvčích (vyšší znalostní úrovně a konverzační kurzy) tak po českých lektorech (začátečnické kurzy).

Jazyková škola, jako ostatně jakákoli instituce netriviální velikosti, musí vykonávat administrativní činnosti. Mezi ty patří zejména sestavování rozvrhu. Počet učeben a časové možnosti jsou omezené, toto je navíc potřeba uvést do souladu s počtem přijatých přihlášek, protože otevírat kurzy pro určité počty studentů se může a nemusí vyplácet, někdy je možné dva kurzy sloučit atd. Další důležitou činností je kontrola fungování lektorů a evidence odučených hodin, s čímž úzce souvisí také evidence přijatých plateb od klientů, výplaty zaměstnancům, pronájem prostor a další finanční úkony. Rovněž lze uvést nábor lektorů, komunikaci s klienty, propagační činnost a jiné, přičemž tyto agendy mohou a nemusí být podporovány informačními technologiemi.

2.3 Lingua Nostra v prvním přiblížení

V první fázi své činnosti jsem se soustředil na definování stavu a fungování Lingua Nostra, abych identifikoval problematické oblasti fungování. Zaměřil jsem se především na enterprise úroveň Lingua Nostra a na aktivity typické pro analýzu na této úrovni: as-is stav, business potřeby a business model [7, p. 46]. Tyto výstupy byly součástí mého semestrálního projektu. Shrňme je tedy znovu zde.

2.3.1 Business model

Business model Lingua nostra byl vytvořen metodikou *Business Canvas* a popisuje obrázek v příloze B.

Klíčovým zákaznickým segmentem jsou pracující lidé středního věku, nezřídka mající rodiny, pro něž je studium italštiny volnočasovou aktivitou a nabyté jazykové dovednosti pro ně nejsou nutně prvořadé. To je v souladu s prioritizací volnočasové funkce oproti výukové. Méně často jsou pak klienty Lingua Nostra studenti se zájmy o jazyky, či firmy, které mají zájem o zvýšení kvalifikace svých zaměstnanců.

Budování loajální zákaznické základny je podporováno kupóny na zážitkové akce pro stálé klienty, či pomocí Open Class, jednorázových výukových lekcí zdarma. Významnou roli zde hraje i osobní komunikace jak s lektory, tak s managementem, která vytváří pocit individuálního přístupu mnohem lépe, než by to dokázaly automatizované e-maily nebo chatbot.

Komunikačními kanály s potenciálními zákazníky je především webová stránka (viz obrázek 2.1), která umožňuje rychlé a přehledné vybírání kurzů či zážitkových akcí a bezproblémové přihlášení na ně, a také e-shop Italské knihy, který prodává nejen učebnice a zjednodušenou četbu, ale i knihy o reáliích, kuchařky, beletrii a další. Publicitu zajišťuje reklama na sociální síti Facebook a stabilní pozice na vrcholu žebříčku vyhledávání na googlu pro heslo "italština Praha."

Obrázek 2.1: Webová stránka Lingua Nostra

Klíčovým přínosem pro zákazníky je právě již zmíněná relaxační a zábavná, nikoli výkonově zaměřená atmosféra výuky. Spíše než certifikáty nebo vysokou úroveň dovedností nabízí Lingua Nostra italštinu a Itálii jako zážitek a odpočinkovou aktivitu. Příjemnou atmosféru podporuje jak již zmíněná osobní komunikace, tak domácky vybavený interiér recepce a učeben.

Z toho vyplývají klíčové aktivity, které klientům tyto přínosy zprostředkovávají. Je to především výuka, kromě pravidelných gramatických kurzů jsou pořádány i jednorázové kurzy zaměřené např. na italská gesta nebo kurzy čistě konverzační. Součástí jsou i kurzy individuální, které často probíhají mimo prostory Lingua Nostra, v některých případech i u klienta. Nezanedbatelnou roli zde hrají doplňkové akce, jako například kurz akvarelové malby v italštině nebo virtuální prohlídky italských měst.

Mezi zdroji, které tyto aktivity umožňují, zaujímá klíčové postavení lektorský tým. Lektoři jsou z velké části rodilí mluvčí pracující na plný úvazek, jsou osvědčení, osobně se znají s dlouhodobými studenty a snadno navazují vztahy s novými. Dále je důležitý administrativní portál, kde probíhají organizační agendy, jako je vypisování, přesouvání a rušení jednotlivých kurzů. Pro tyto a další funkce jsou nezbytní i zkušení administrativní pracovníci, kteří mají dobré znalosti interního fungování i lektorů, a proto jsou schopni zajišťovat bezproblémově náhrady lekcí, suplování, komunikaci se studenty a jiné.

Nejdůležitějšími partnery Lingua Nostra jsou italské univerzity, které do České republiky vysílají stážisty, což jsou obvykle studenti didaktiky italštiny, pro které je stáž v Lingua Nostra cennou zkušeností. Stážisté v rámci své stáže vyučují jako lektoři skupinových a individuálních kurzů, čímž získávají cenné pedagogické zkušenosti.

Struktura nákladů je relativně prostá. Jednu část tvoří mzdy zaměstnanců (tedy lektorů a administrativních pracovníků) a druhou pak pronájmy za prostory, kde výuka probíhá. Výnosy jsou pak platby za všechny typy kurzů a zážitkové akce, v menší míře pak tržby z e-shopu a z tlumočení a překladů.

2.3.2 Organizace

Nyní, když máme zjednodušeně popsaný business model, podívejme se o úroveň níže a ptejme se, jak je fungování byznys modelu zajištěno. Nejprve popíšeme organizaci Lingua Nostra z hlediska aktérů, které má k dispozici a vyjmenujeme klíčové procesy, jichž se tito aktéři účastní.

Předně má Lingua Nostra vedoucí, která je majitelkou a má na starosti řídící procesy a do určité míry administrativní agendy. Kromě těch agend, jimž se věnuje spolu s ostatními administrativními pracovníky, je jejím úkolem vypisování kurzů a organizace doplňkových akcí.

Administrativním agendám se rovněž věnují dva administrativní pracovníci, kteří omezeně působí i jako lektoři. Administrativní agendy, které tvoří hlavní část jejich činnosti, obnášejí komunikaci s klienty, zajišťování náhradních lekcí a suplování, nábor lektorů a stážistů, plánování rozvrhů a organizaci didaktiky, v neposlední řadě také rušení lekcí, žádají-li si to okolnosti. Ostatní lektoři se věnují především výuce a na starost mají jen ty administrativní agendy, které se přímo týkají jejich výuky, jmenovitě vyplňování měsíčních

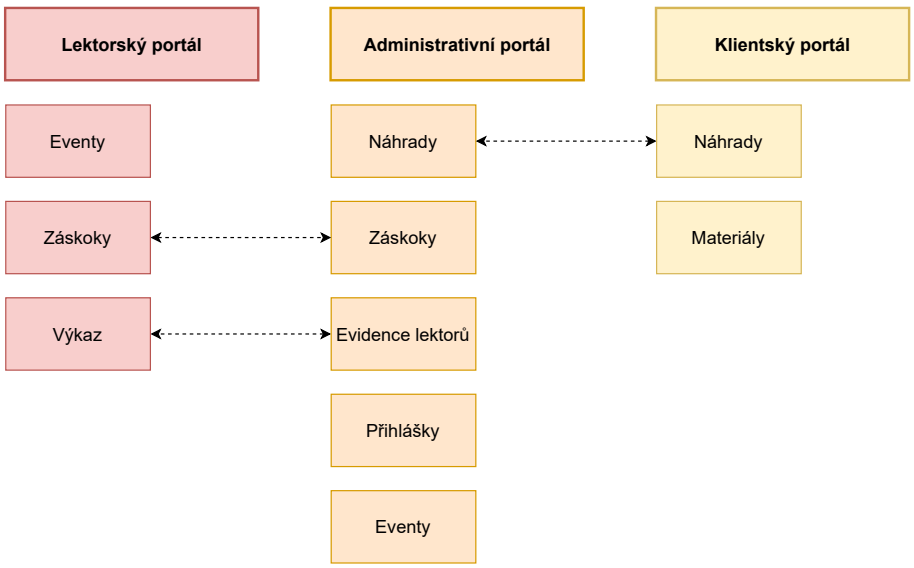
výkazů a v případě potřeby suplování. IT pracovník má na starost činnosti související s údržbou stránek, jejichž součástí je i administrativní portál, a jejich vývoj.

Do procesů vstupují i samotní klienti - komunikují, provádějí platby, účastní se výuky i akcí a žádají o náhrady lekcí. Platba má různé účely: skupinový kurz, doplňková akce nebo jednorázový kurz. Speciální režim pak mají individuální studenti, protože mohou kupovat balíček nebo platit fakturou po měsíci.

Je nutné také zmínit časové hledisko fungování Lingua Nostra. Rok se z hlediska výuky dělí na tři období: podzim, jaro a léto. Skupinové kurzy typicky trvají jedno období. Když období skončí, studenti se přesunou do navazujícího skupinového kurzu odpovídajícího vyšší úrovni znalostí a do uprázdněného kurzu se přesunou studenti z kurzu nižšího. Podzim je nejdůležitějším obdobím, protože právě zde se přihlašuje a začíná nejvíc nových studentů. Důležité je také jaro, protože spolu s podzimem tvoří tu část roku, kdy probíhají skupinové kurzy, které jsou velkým zdrojem příjmů. Naopak v létě probíhá výuky nejméně a vždy dochází k propadu příjmů, který se však v posledních letech znatelně snížil.

2.4 Stav podpůrných systémů

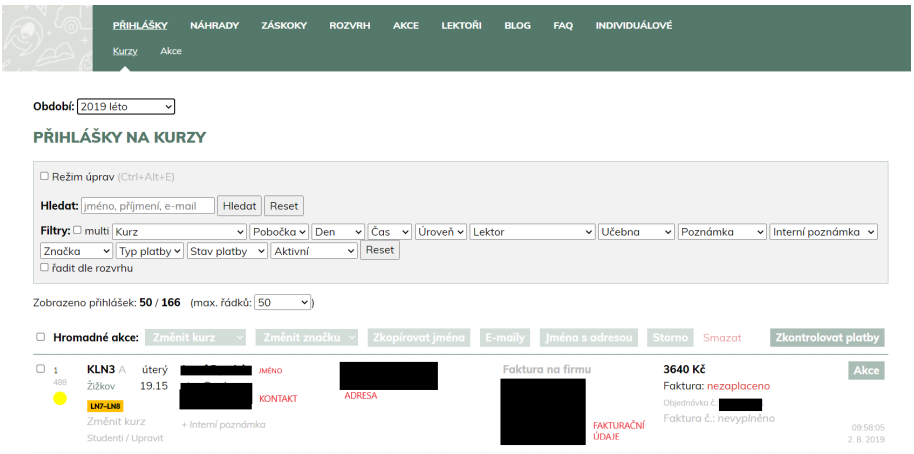
V současnosti Lingua Nostra používá jeden, svépomocí vyvinutý, informační systém. Jedná se o webovou aplikaci napsanou v PHP, nepoužívající žádné frameworky, která zároveň slouží jako webová stránka Lingua Nostra. Pro přístup k funkcionalitám informačního systému je nutné přihlášení. Tyto funkcionality jsou seskupeny do tří „portálů:“ administrativního, lektorského a studentského. Popišme nyní stručně, jaké agendy jednotlivé portály podporují. Zjednodušenou strukturu portálů a funkčností, které poskytují vidíme na následujícím obrázku 2.2:



Obrázek 2.2: Administrativní portál

2.4.1 Administrativní portál

Hlavní agendou administrativního portálu (který vidíme na následujícím obrázku) je evidence přihlášek na kurzy. Kvůli této agendě portál historicky vznikl, protože s rostoucím počtem studentů bylo velmi obtížné udržet si přehled o tom, které přihlášky jsou zaplacené, a které nikoli. Studenti nyní vyplňují přihlášky online a portál je propojen se službou Fakturoid. Tato služba je cílená na podnikatele a umožňuje jim efektivní a přehlednou správu účetnictví, včetně evidence faktur. Vyplní-li student přihlášku, je mu vystavena faktura. Jakmile je zaplacena, fakturoid ji jako takovou zaeviduje a oznámí to aplikaci Lingua Nostra pomocí webhooku. Zjednodušeně řečeno, místo toho, aby např. v našem konkrétním případě Lingua Nostra musela v pravidelných intervalech dotazovat API Fakturoidu, bude Fakturoid odesílat oznámení o změnách na námi zadané url. Zde je třeba poznamenat, že typy upozornění, které Fakturoid umí posílat, jsou velmi omezené a v budoucnu se to může pro stávající systém ukázat jako výrazný problém.



Obrázek 2.3: Administrativní portál

Přihlášky je možné filtrovat podle mnoha různých kriterií, například full-textovým vyhledáváním podle jména, příjmení nebo e-mailu, kurzu, na který se klient přihlašuje, interní poznámky, nebo typu platby. Mimoto umožňuje tento seznam přihlášek u jednotlivých položek například změnit kurz, což je potřeba v případě, kdy se student hlásí na kurz, který byl z nějakého důvodu zrušen a je potřeba jej přesunout jinam. Dále je možné stornovat platby nebo zkopírovat e-maily zadavatelů všech vybraných přihlášek do clipboardu (díky tomu je lze následně vkládat pomocí ctrl+v). To usnadňuje posílání hromadných e-mailů. Analogickým způsobem jsou evidovány ne-kurzovní akce. Jedná se o tematické aktivity, jako například zájezdy, výlety a prohlídky zaměřené na italštinu nebo italskou kulturu.

Dalším důležitým aspektem jsou náhrady. Lingua Nostra umožňuje studentům si nahradit zaplacené lekce, na které se z nějakého důvodu nemohli dostavit, na jiném kurzu (v jiném termínu a s jiným lektorem). Přesná evidence těchto náhrad je potřebná také z toho důvodu, že kurzy mají maximální

kapacitu. Jednak kvůli velikosti učeben a jednak kvůli kvalitě výuky. Není tedy možné automaticky přesunout studenta na jakýkoli vybraný kurz, ale pouze na ten, kde je volné místo. Je navíc třeba předem uvědomit lektora.

Analogicky fungují záskoky. Jedná se o suplování, tedy situaci, kde lektor pravidelně učí nějaký kurz, ale nemůže se na výuku dostavit. Například kvůli nemoci nebo jiným důvodům. Pro klienty by bylo velmi nepříjemné výuku v takovém případě rušit, třeba i s vrácením peněz, a proto se Lingua Nostra vždy pokouší najít náhradního lektora. To není vždy snadné, protože časové možnosti lektorů mimo kurzy, které učí, jsou nepravidelné. Tato funkcionalita informačního systému pouze eviduje žádosti a konkrétního náhradního lektora hledá administrativní pracovník s použitím své mimosystémové evidence a osobních znalostí.

Administrativní portál umožňuje také zobrazení rozvrhu, a sice ve dvou pohledech. První pohled je seznamový, kde aministrátor vidí aktuálně probíhající kurzy s informacemi o nich, přidává nové, může je řadit, filtrovat, měnit u nich lektora a podobně. Druhý pohled je grafický, zobrazuje strukturu týdne pro vybranou místnost, s kurzy v časech, kdy se učí. Tento pohled však má jedenapůlhodinovou granularitu a není tudíž vhodný pro evidenci jakéhokoli jiného využití místností.

Nedávno byla implementovaná nová funkcionalita přidávání mimovýukových akcí, které dosud musely být přidávány manuálně do databáze. Akce mohou také být filtrovány podobně jako přihlášky. Pro přehlednost dodejme, že na rozdíl od řazení přihlášek na akce, kde je hlavním cílem poskytnout přehled o tom, kdo je přihlášen a kdo zaplatil, sděluje tato funkcionalita administrátorovi, jaké akce jsou vypsané a kdy a kde se konají. Jedná se mimochodem o typický příklad rozrůstání systému v souvislosti se změnami business modelu LN. Původně se mimovýukové akce nekonaly, pak jen v omezené míře a jejich evidence tudíž byla jednoduchá, v poslední době ale tvoří významnou část produktu Lingua Nostra. Proto byla implementována jejich evidence v rámci stávajícího systému. Díky ní je dokáží administrativní pracovníci efektivně spravovat navzdory jejich velkému množství.

Poslední důležitou částí je evidence lektorů. Zobrazují se zde částky, které je třeba vyplatit lektorům za výuku v uplynulém měsíci, přílohy výkazů (typicky scany papírových dokumentů, kde studenti potvrzují docházku) a je zde také možné nastavovat lektorům finanční bonusy v jednotlivých měsících.

2.4.2 Lektorský portál a klientský portál

S lektorským portálem zacházejí lektoři, vyučující - klíčoví zaměstnanci LN. Hlavní funkcionalitou jsou zde výkazy. Tyto lektor vyplňuje vždy na konci měsíce nebo na začátku následujícího. Automaticky se mu načtou kurzy, které vyučuje a on pak zadá počet hodin, které v tomto měsíci tento kurz učil. Mimoto zde zadává manuálně individuální výuku a vedení mimovýukových akcí.

Mimoto umožňuje tato sekce, aby lektoři žádali o suplování. Zde se vyplňuje konkrétní kurz, datum a čas a poznámky ohledně látky, která se měla na dané hodině probírat. Tato funkcionalita má svůj protipól v administrativním

portálu. Zejména vykazování individuálních studentů je (i podle názoru vedení) nedostatečná, protože se zde pouze volně zadává jméno studenta, počet lekcí a cena. Systematičtější by bylo, kdyby byli v současnosti aktivní individuální studenti evidováni a lektor je pouze vybíral ze seznamu, přičemž by se automaticky načetly jejich ceny za lekci a on by pouze doplňoval jejich data.

Dále je v tomto portálu, podobně jako v administrativním, implementován přehled akcí. Jsou seskupeny podle měsíců a poskytují informace o názvu, čase, místu konání a počtu přihlášených studentů.

Klientský portál obsahuje logiky ze všech nejméně. Jedním z důvodů je to, že některé důležité agendy (jako například podávání přihlášek) se nedějí v portálu se zabezpečeným přístupem, ale v částech aplikace dostupných veřejnosti otevřeně jako webová stránka. Z funkcionalit probíhajících v portálu je třeba zmínit především žádosti o náhradní lekce. Tato funkcionalita není příliš často využívána (tento údaj pochází od administrativních pracovníků), přesto ale v případech, kdy na její využití dojde, výrazně urychluje komunikaci. Klient žádající o náhradní lekci musí vyplnit alespoň dva možné kurzy, o něž by měl zájem jako o náhradní. Administrativní pracovník pak ve svém portálu zkontroluje, zda je možné tam studenta přesunout a oznámí mu, kam jej zařadil. Pokud žádný kurz nevyhovuje, požádá administrativní pracovník klienta o další termíny.

Účelem této kapitoly bylo pochopit fungování Lingua Nostra do té míry, aby bylo možné navrhnout některé kandidáty na oblasti činností, které je vhodné podpořit rozšířením stávajícího systému. Zmapoval jsem procesy, které zde probíhají a popsal jsem strukturu informačního portálu, který je podporuje. V další kapitole se již začneme věnovat možným rozšířením.

Kapitola 3

Oblasti fungování stávajícího systému vhodné pro rozšíření podpory

Nyní již známe stávající systém dobře, přinejmenším z uživatelského pohledu, domnívám se tedy, že můžeme přistoupit k zásadnímu bodu práce, totiž otázce, co mohu dodat, aby to pro Lingua Nostra bylo významným přínosem. V této kapitole se budu věnovat rozboru jednotlivých možností rozšíření stávajícího systému, které vzešly z konzultací s vedením Lingua Nostra.

Stávající systém se dynamicky vyvíjí a stále jsou do něj přidávány nové funkcionality, například výše zmíněné vypisování mimovýukových akcí. V minulosti přidané funkcionality často ušetřily značné množství práce, snížily chybovost a v konečném důsledku zvýšily výnosy, především propojení s platební bránou a systém přihlášek. Stávající systém vyvíjel od začátku stejný IT pracovník, který se s vedoucí dobře zná. Proto má přehled jak o systému, tak o fungování Lingua Nostra. To mu výrazně ulehčuje implementaci nových funkcionalit, když se ukáží jako potřebné. Protože mi nabídl součinnost a své znalosti, zaujal jsem stanovisko, že rozšíření stávajícího systému je schůdná varianta.

Spolu s vedoucí jsem prodiskutoval různé možnosti, jak systém účelně rozšířit. Tato debata byla obsáhlá a plodná, proto bych ji zde rád rekapituloval a uvedl argumenty, proč jsme některé nápady opustili a jiných se naopak drželi, a k jakému rozhodnutí jsme nakonec došli.

Předně je třeba zmínit, jak jsme se postavili k možnosti stávající systém opustit a vyvinout zcela nový. Tento návrh musel nevyhnutelně padnout a některé argumenty v jeho prospěch byly přesvědčivé. Oproti stávajícímu systému bychom mohli přepracováním procesu přihlášení zlepšit bezpečnost, celkově by bylo možné zvýšit robustnost a modularitu. Stávající systém navíc postrádá jasně definovanou architekturu (blíže viz kapitola Řešení, sekce Architektura) a nedostatek robustnosti je palčivý: nové funkcionality se často dostávají do konfliktu se stávajícím kódem a jejich implementace je stále složitější.

Tento návrh jsme však nakonec zavrhli. Důvodem bylo, že tento systém již implementuje mnoho hojně užívaných funkcionalit a jeho vývoj nadále probíhá. Dalším vážným problémem se zcela novým systémem by mohla být migrace dat, v jejichž ukládání nepanuje v současnosti úplná konzistence. Rozhodli jsme se tedy definitivně pro rozšíření stávajícího systému. Osobně se

ale domnívám, že v budoucnosti bude muset dojít buď k zahození stávajícího systému nebo jeho rozsáhlému refactoringu, protože vývoj tak, jak probíhá v současnosti, není udržitelný. Při hledání vhodných rozšíření jsme se soustředili na taková, která odstraní nějaký naléhavý problém a zároveň nebudou triviální.

Jedno z rozšíření, které vyplynulo jako potřebné téměř okamžitě, byla evidence individuálních kurzů. Tato agenda odčerpává mnoho energie administrativních pracovníků a vlastně i lektorů. Její implementaci usnadní i to, že je již dlouho plánovaná a existuje pro ni dobře dokumentovaný návrh na interním Trellu. Implementaci této funkcionality, relativně přímočaré a přitom velmi přínosné, jsme se rozhodli zařadit jako tzv. quick-win.

Dále jsme hledali rozsáhlejší oblast, kterou by bylo možno podpořit, třeba i programátorsky náročnější a jejíž přínos nebude nutně okamžitý. V souvislosti se stále ještě probíhající pandemií se naše myšlenky zaměřily na podporu online výuky, konkrétně na rozšíření studentského portálu o interaktivní výukové prvky. Programování výukových her nebo e-learningu by sice rozhodně bylo zajímavé, avšak v tuto chvíli pro to chybí pedagogická koncepce a není ani jasné, zda by o takovouto funkcionalitu klienti vůbec měli zájem. Tuto možnost jsme tedy odsunuli do oblasti návrhů možného dalšího směřování.

Následující možnost - automatizace záskoků - byla blíže aktuálním problémům fungování Lingua Nostra a její přínos se mnohem lépe vyhodnocoval. Centralizace a zpřehlednění informací o tom, kdo kdy může suplovat, proces záskoků zpřehlední a převede do systémovější podoby. Další výhodou bude, že pak nebude tolik závislý na osobních znalostech administrativních pracovníků. Proti však mluví samotná povaha těchto informací: jsou nesourodé a často a rychle se mění, je tedy otázkou, do jaké míry vůbec je možné dosáhnout větší přehlednosti, než jakou má jedním člověkem spravovaný wordovský dokument. Mimoto nejsou časové ztráty v tomto procesu nijak drastické a neprobíhá příliš často, v důsledku čehož jsme zavrhli i tuto možnost.

Oproti tomu vyřizování e-mailů od klientů je činností nesmírně časově náročnou a prakticky nepřetržitou. Pokud se podaří v těchto e-mailech vystopovat nějakou pravidelnost a namodelovat automatické odpovědi, které klientům poskytnou požadované informace, bude úspora času obrovská. Došli jsme však k tomu, že tento klíčový předpoklad naplnit nedokážeme. Z osobních zkušeností vedoucí vyplývá jednak, že v e-mailech od zákazníků takové vzorce, na které je možné napasovat univerzální odpověď, zkrátka nejsou a co víc, mnohdy se jedná o děkovné e-maily spokojených zákazníků, kteří se chtějí podělit o svou spokojenost. V takovémto případě je strojová a chladná automatická odpověď vysloveně kontraproduktivní. Osobní komunikace je podstatným pilířem byznys modelu Lingua Nostra a tato automatizace jej podkopá.

O automatizaci rušení lekcí jsme se zmínili jen letmo a prakticky hned jsme se shodli na tom, že to není vhodná oblast pro rozšíření. Problémy se zrušením lekce způsobuje to, že zrušení lekce předpokládá její náhradu, a to je realizováno faktickým prodloužením trvání kurzu o týden, což může například v podzimním období vést k tomu, že kurz zasáhne buď do vánočních

svátků, nebo hůř, do začátku dalšího období. Toto nastává hlavně v případech, kdy hodina odpadne opakovaně. Argumentem proti je zde zřídkavost této situace. Ze zkušeností administrativních pracovníků vyplývá, že k situaci, kdy je posun kurzu tak problematický, že jej nelze vyřešit osobní domluvou s klienty v kurzu, prakticky nedochází.

O poznání slibnější se jevila oblast přihlašování a odhlašování. Určitý mechanismus zajišťující autorizaci klientů, lektorů a administrativních pracovníků implementován sice je, avšak míra zabezpečení, kterou poskytuje, je ve srovnání s obvyklejšími standardy podle mého názoru nižší. Jsou zde také některé mezery v autentizaci lektorů a určité části procesů, které lektorský portál zahrnují, spoléhají na vzájemnou důvěru lektorů a administrativních pracovníků. Tento způsob přihlašování a odhlašování však dosud funguje bezproblémově a nezdá se, že by např. slabší autentizace lektorů způsobovala komplikace nebo dokonce pokles výnosů. Rozhodli jsme se tedy overhaul přihlašování alespoň pro tuto chvíli opustit.

Kvůli pandemii do určité míry poklesl zájem studentů, v důsledku čehož výrazně kleslo využití prostor. V souvislosti s tím vznikl nápad, najít pro ně alternativní využití, konkrétně je pronajmout. Pronájem bude do určité míry kompenzovat propad příjmů, a po návratu do normálu je v součtu může zvýšit.

Toto rozšíření představuje úplně novou sadu funkcionalit a má značný přesah do dalších oblastí. Vyžaduje spolehlivou evidence a na ni navázané zajištění plateb. Využití místností potřebuje nějakou formu dokumentace, ideálně rozvrhu, který umožňuje rezervovat místnosti dopředu a s určitou granularitou. Tento rozvrh musí umět zohlednit dny a hodiny již vypsaných kurzů. V neposlední řadě musí být nějak zajištěna ochrana informací - potenciální nájemník musí vědět, kdy jsou dané místnosti volné, ale informace o tom, jaký kurz se kterým lektorem a kterými studenty se kde koná, mu nemají být přístupné.

Jedná se tedy o oblast rozsáhlou, jejíž přínos bude dobře vyčíslitelný, i když patrný až po nějaké době. Implementace obnášela důkladné seznámení se strukturou dat a úpravu databázového schématu, zmapování stávajícího kódu, (protože některé z již implementovaných funkcí bude možné použít) i vývoj nové grafické komponenty. Spolu s vedením jsme se shodli na tom, že tato oblast je ideálním kandidátem na rozšíření stávajícího systému.

V souvislosti s touto oblastí jsme se rozhodli přidat také grafický rozvrh do portálu lektorů, jimž umožní rezervovat místnosti pro individuální výuku, a do administrativního portálu. Tam se sice již jeden rozvrh nachází, avšak ten zobrazuje pouze pravidelné kurzy v dané učebně. Neumožňuje administrativním pracovníkům vytvářet či rušit rezervace a nezohledňuje individuální výuku. Je také otázkou, zda by jeho granularita vyhovovala i pro pronájmy a jak do něj bude možné data o pronájmech integrovat. Nepřímým přínosem tohoto rozvrhu bude i určité zjednodušení dalších agend, jako je například nahrazování individuálních lekcí: administrativní pracovník okamžitě uvidí, kterou učebnu je kdy možné poskytnout pro náhradní výuku.

Kapitola 4

Vybraná rozšíření stávajícího systému

V předchozí kapitole jsme se věnovali diskuzi možných rozšiřujících funkcionalit a argumentům, proč jsme některé z nich zavrhli a jiné přijali. Přesný popis těch funkcionalit, pro které jsme se nakonec rozhodli, jsme však dosud neprovedli, proto jej uveďme zde. Aby je bylo možné implementovat, musíme přesně vědět, co od nich očekáváme. Popis těchto očekávání je obsahem této kapitoly. Pro přehlednost je rozděluji do tří větších bloků: Evidence individuálních studentů, Pronájmy a Rozšíření lektorského portálu.

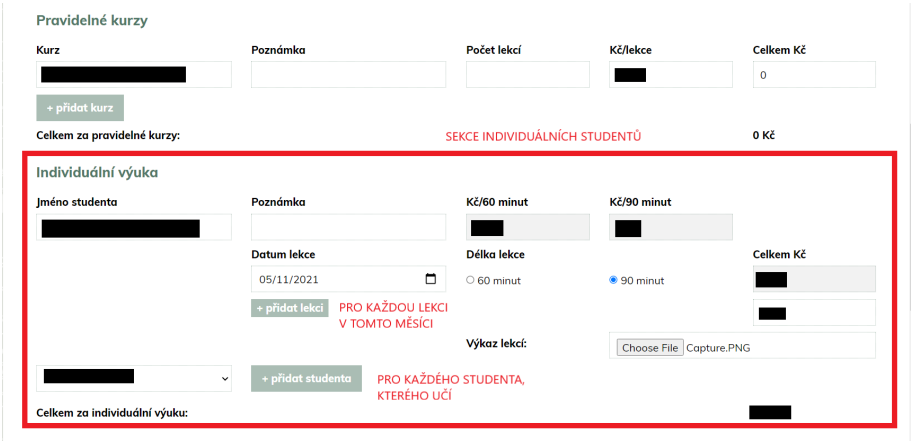
4.1 Evidence individuálních studentů

Jak již bylo řečeno, evidence individuálních studentů probíhá, avšak od evidence skupinových kurzů se značně liší svou nesystémovostí a náročností. Jedná se o několik značně komplikovaných excelových tabulek. Implementace tedy musí umožnit individuální studenty evidovat přesně, systematicky a šetřit zdroje. Základním prvkem bude zavedení individuálního studenta jako systémové entity. Kromě identifikačních údajů jako je jméno, příjmení, telefonní číslo a e-mailová adresa budeme zaznamenávat také metodu platby, která je v Lingua Nostra dvojí. Jednak fakturou, kdy je studentovi posílána na konci každého měsíce faktura za všechny lekce, které tento měsíc měl, a jednak balíčkem: klient dopředu zaplatí dohodnutou částku a z té se mu pak strhávají ceny odučených lekcí. Dalšími informacemi jsou pak cena za 60 a 90 minut výuky, zbývající kredit (v případě individuálních studentů platících balíčkem) a aktivní status - individuální studenty, kteří v současné době žádnou výuku neplatí, bude možné ponechat v systému, ale podle této položky je filtrovat. Tyto informace musí být dostupné administrátorovi a ten je také musí být schopen měnit.

Dalším zásadním prvkem, který jako prerekvizitu vyžaduje existenci studenta jako entity, bude přehled individuálních studentů a pohledávek vůči nim, podobným způsobem, jako v současné době funguje evidence faktur za skupinové kurzy. Konkrétně bude v seznamu studentů vidět, zda daný student platí balíčkem či fakturou a jaký je stav jeho kreditu. Ten bude v případě studentů platících fakturou vždy nekladný, neboť tito studenti na konci měsíce hradí jen tu výuku, kterou již měli, přeplatky u nich tedy nevzniknou. Oproti tomu u studentů platících balíčkem je kladný kredit obvyklý a z podstaty

věci žádoucí, naopak záporný kredit může vzniknout tak, že student vyčerpá víc lekcí, než na kolik stačila jeho balíčková platba. V takovém případě bude administrátor moci vystavit fakturu na další balíček. Fakturace, i u studentů, kteří balíček nepoužívají, bude probíhat přes Fakturoid, jako je tomu i v případě akcí a skupinové výuky. Je tedy nutné zajistit propojení s touto službou. U těch, které již byly vystaveny, se zobrazí číslo faktury.

Třetí prvek evidence individuálních studentů pracnost nesníží, zato však sníží chybovost. Jedná se o úpravu stávajícího formuláře měsíčních výkazů lektorů tak, aby umožňoval lektorům přidat do výkazu všechny individuální studenty, s nimiž daný lektor v tomto měsíci měl výuku. Lektor dále manuálně přidá všechny lekce, které daný měsíc proběhly, vybere u nich datum, nastaví délku 60 nebo 90 minut a následně nahraje fotku nebo scan výkazu. Toto rozšíření vidíme na následujícím obrázku:



Obrázek 4.1: Rozšíření lektorského výkazu

4.2 Nájemní portál

Pronájmy místností externím subjektům jsou nejrozsáhlejší rozšiřující funkcionalitou. Využití prostor se jimi diverzifikuje a značně znepřehlední, je tedy potřeba implementovat nový způsob evidence rozvrhu a práce s ním. Nájemníkům bude administrátor manuálně vytvářet účty, pomocí nichž budou nájemníci interagovat s LN. Autentikace nájemníků zpřehlední komunikaci a zajistí možnost propojení využití místností nájemníkem a následnou výší požadované platby.

Tyto interakce budou obnášet rezervace půlhodinových slotů v konkrétní místnosti na konkrétní dobu, jejich případné rušení a následnou fakturaci ze strany Lingua Nostra za tyto pronájmy. Aby nájemník mohl sloty rezervovat, musí vidět, které z nich jsou volné a které nikoli. To s sebou ovšem nese i nutnost určité ochrany dat: ty sloty, které volné nejsou, protože v nich probíhá výuka či akce Lingua Nostra nebo je má pronajaté jiný nájemník, se musí zobrazovat pouze jako nedostupné a nezobrazovat další informace. Nájemník musí tedy být schopen rozlišit pouze sloty volné, obsazené, a rezervované

sebou - ty pak bude moci za určitých podmínek zrušit.

Rušení rezervovaných slotů je žádoucí, protože v případě, že je nájemník nebude moci využít, nebudou muset zůstat prázdné a bude do nich možné např. přesunout některou náhradní individuální výuku. Rušení slotů však musí mít i svá omezení, zaprvé nesmí být možné zrušit sloty, které jsou již v minulosti - jinak by si nájemci mohli uměle snižovat náklady - a zadruhé musí být možné rušit jen ty sloty, do jejichž začátku zbývá více než 24 hodin, jinak pro ně nemůže Lingua Nostra realisticky stihnout najít využití.

Vhodným řešením, které tyto funkcionality implementuje, bude nová forma rozvrhu. V administrátorském portálu již rozvrh existuje, má však odlišnou granularitu, umožňuje filtrovat po budovách, ale ne po učebnách, nezohledňuje akce a individuální výuku a předpokládá stejný rozvrh pro všechny týdny období. To není pro potřeby pronájmů dostatečně flexibilní. Vytvořil jsem tedy nový, s půlhodinovou granularitou, podporující jak zobrazení vypsaných kurzů, tak akcí a výuky individuálních studentů. Tento kalendář umožňuje označit jeden nebo více slotů a rezervovat je. Rezervované sloty tak administrátor a lektoři uvidí ve svém kalendáři jako zaplněné.

4.3 Rozšíření administrativního portálu

Díky tomu, že všechny proběhlé pronájmy jsou evidovány v rozvrhu zmíněném v předchozím odstavci, je možná i fakturace nájmů z administrátorského portálu. Nová fakturační komponenta funguje bude fungovat podobně jako u individuálních studentů, konkrétně zobrazuje seznam nájemníků a z rozvrhu vypočítá částku, kterou by v daném měsíci měli nájemníci zaplatit. Umožňuje vystavit klientovi za konkrétní měsíc fakturu a sleduje její stav: žádná faktura není vystavená, faktura je vystavená, faktura je zaplacená.

Pokud ale má tento nový rozvrh poskytovat všechny potřebné informace, nemůže být dostupný pouze nájemníkovi. Musí být dostupný přinejmenším také administrativnímu pracovníkovi, a to z vícero důvodů. Administrátor jednak musí mít možnost zadávat do rozvrhu akce, pokud se konají v prostorách LN, aby nedocházelo ke kolizím například s pronájmy. Dalším důvodem je, že administrátor musí mít o pronájmech (jakož i o ostatních činnostech, které v učebnách probíhají), přehled, z čehož plyne, že administrátor musí vidět informace o všech slotech, jak interních, tak rezervovaných všemi nájemníky. Je také žádoucí umožnit administrátorovi sloty rušit v případě, že to bude potřeba a z nějakého důvodu tak nebude moci učinit nájemník.

S tím souvisí i to, že administrátor potřebuje mít o nájemnících i obecnější přehled, než zobrazení rezervovaných slotů v rozvrhu. Přidal jsem tedy další sekci administrátorského portálu, která umožňuje zobrazit všechny zaregistrované nájemníky a jejich rezervované sloty v seznamovém režimu - tedy pohled, kdy administrátor uvidí všechny sloty daného nájemníka pohromadě, aniž by je musel jednotlivě hledat v rozvrhu. Dále má administrátor možnost zrušit nájemníkovi účet, pokud se spolupráce s ním stane problematickou. Zrušení účtu způsobí, že se všechny nájemníkovy sloty zruší a nebude se již moci dále přihlásit.

Přehled nájemníků administrátorovi také umožňuje upravovat cenu jednotlivých místností pro konkrétního nájemníka. Tímto je umožněno například poskytování slev v případě rezervování více slotů najednou. Díky tomu, že do rozvrhu má přístup administrátor a nájemník, bude rozvrh schopen zohlednit podstatnou část informací o využití místností, zbývají však ještě dvě další oblasti: individuální studenti a pravidelné výuky. Je možné, aby je zadával administrátor, bude to však náročné na zdroje a zvýší to chybovost. Proto se pravidelné výuky budou propisovat z vypsaných kurzů a individuální výuku budou zapisovat lektoři.

4.4 Rozšíření lektorského portálu

Změny v lektorském portálu jsou tedy dvě: za prvé změna vyplňování měsíčního výkazu (přibyla sekce pro individuální studenty) a za druhé přidání rozvrhu, podobného tomu administrativnímu a nájemnickému.

Zde se vrací otázka ochrany dat: na rozdíl od administrátora není záhodno, aby lektor viděl informace o pronájmech. Avšak hlavní funkcí, kterou bude rozvrh lektorovi poskytovat, bude výše zmíněné rezervování místností pro individuální výuku, musí tedy vidět, které sloty jsou volné a které ne. Přijatelným kompromisem je, aby viděl všechny sloty rezervované nájemníkem jako rezervované, bez dalších informací. Naopak o interně rezervovaných slotech lektor informace uvidí, neboť je to žádoucí pro komunikaci a organizaci.

Tato kapitola přiblížila účel a podrobnější vlastnosti rozšíření, které jsem se na základě konzultací s vedením Lingua Nostra rozhodl implementovat. Je zřejmé, že se nejedná o rozšíření zcela triviální. Je třeba zohlednit konzistenci dat, bezpečnost informací a uživatelskou přívětivost. V následující kapitole se budu věnovat tomu, jak jsem tyto funkcionality implementoval.

Kapitola 5

Popis řešení rozšiřujících funkcionalit

V této kapitole přiblížím technickou stránku implementovaných funkcionalit. Nejprve se budu věnovat výběru architektury a jeho zdůvodnění. Konkrétně se bude jednat o vrstevnatou architekturu a architekturu MCV. V části Implementace rozšiřujících funkcionalit pak budu popisovat některé vybrané aspekty řešení podle jednotlivých segmentů vrstevnaté architektury, počínaje perzistenční vrstvou, přes vrstvu business logiky a konče prezentační vrstvou, s níž úzce souvisí problematika controllerů. Následně se přesunu k bezpečnostním aspektům nově implementovaných funkcionalit. Budu se zabývat hashováním hesel a udržování session pomocí JSON Web Tokenu. V závěrečné části kapitoly se budu věnovat testování nových funkcionalit, konkrétně jednotkovému a uživatelskému.

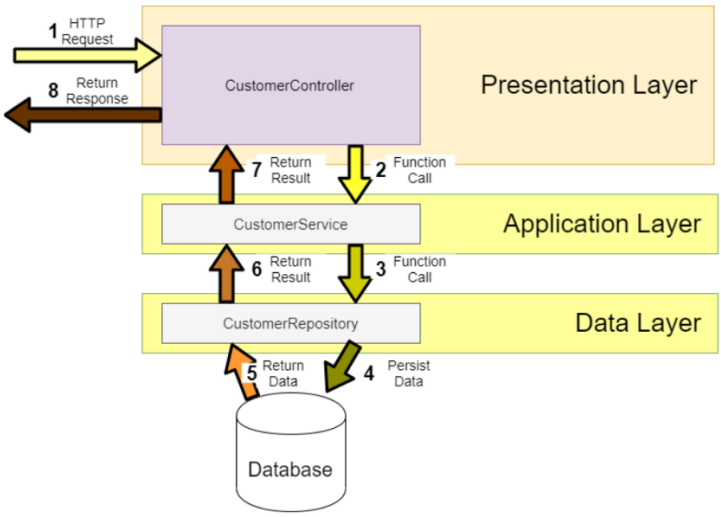
5.1 Softwarová architektura rozšiřujících funkcionalit

V kódu stávajícího systému není patrná žádná konkrétní softwarová architektura, což je nedostatek, neboť architektura v softwaru plní mnoho důležitých účelů, slouží pro pochopení a evoluci myšlenek, dorozumění mezi vývojáři (které v počátečních fázích tohoto projektu místy scházelo a právě jasná architektura by to byla mohla napravit), je svým způsobem formou dokumentace a usnadňuje simulaci a validaci. Volba vhodné architektury před začátkem implementace je důležitá, zde se však potýkáme s poněkud jiným problémem: se systémem již rozsáhlým, plnícím řadu rozličných funkcionalit, který se právě kvůli své rozsáhlosti a zároveň potřebě růstu pomalu stává neudržitelným.

5.1.1 Architektura pro nové funkcionality v administrativním a lektorském portálu

Restrukturalizovat celý projekt dalece překračuje jak rámec této práce, tak mé časové a mentální možnosti. Nezbývá tedy, než zvolit některý architekturální vzor tak, aby alespoň přibližně odpovídal struktuře stávajícího kódu a podle tohoto vzoru implementovat nové funkcionality, čímž bude nastíněna jedna možná osnova pro případný další refactoring funkcionalit stávajících.

Konkrétně jsem se rozhodl pro vrstevnatou architekturu (pojmy „vrstva“ a „stupeň“ používám ve stejném významu jako Martin Fowler, tzn. „stupeň“ implikuje oddělenost hardwaru), a sice proto, že jako nejjednodušší a nejpřehlednější přístup se jeví vzít současné Services (toto označení používá legacy kód, ale třídy, které jej nesou, neodpovídají vrstvě Service ve vrstevnaté architektuře), které obsahují jak prezentační, tak databázovou, tak aplikační logiku a tyto tři aspekty v nich oddělit do klasických tří vrstev: prezentační, business a persistenční. Zde je třeba poznamenat, že tento přístup jsem zvolil pro funkcionality kalendáře a individuálních studentů. Pro funkcionality pronájmů budov jsem zaujal odlišné stanovisko, viz následující subsekce. Následující obrázek ilustruje strukturu trojvrstvé architektury. Podle něj budeme postupovat v sekci Implementace.



Obrázek 5.1: Trojvrstvá architektura, zdroj [11]

Pokud vytváříme webovou aplikaci v PHP, vyvstává často otázka, zda business vrstvu, a potažmo doménový model, vůbec potřebujeme, a pokud ano, pak do jaké míry. Podstatou toho, co webové aplikace dělají, je často v zásadě provádění CRUD operací s entitami a jejich prezentace uživateli přehledným grafickým způsobem. V takovém případě není doménový model příliš potřebný, protože doménový objekt lze nahradit např. asociativním polem. Doménový objekt tedy není o mnoho víc, než obyčejné rozhraní k databázové tabulce [4, p. 120]. Z tohoto důvodu se může jevit jako relativně nepotřebná i business vrstva. Argumentem pro business vrstvu a doménový model je to, že aplikace mají tendenci s časem růst a stávat se složitějšími - nejinak je tomu v té naší - a proto se v budoucnosti může objektová reprezentace modelu ukázat jako užitečná. Já osobně jsem toho názoru, že další přidanou hodnotou doménového modelu je lepší uchopitelnost a představitelnost fungování kódu.

Zásadní přínos zavedení architektury zde vidím v tom, že se mi podařilo dosáhnout oddělení prezentační logiky (ač, jak uvidíme později, ne docela a

ne úplně šťastným způsobem). Oddělení prezentační a doménové logiky je základní princip návrhu softwaru [4, p. 323] a je relevantní nejen pro webové aplikace, ale pro veškerý software, který interaguje s člověkem. Důvodů pro toto oddělení je mnoho, uveďme však alespoň ty základní. Především se jedná o *Separation of concerns*, protože v oblasti uživatelského rozhraní platí odlišná pravidla, používají se odlišné techniky a aplikují se jiné způsoby uvažování. UI se mimoto velmi často mění, a proto je dobré, aby se tyto změny mohly odehrát bez dopadu na doménový kód.

Dalším důvodem je zvýšení přepoužitelnosti: různá uživatelská rozhraní často potřebují pracovat se stejnou doménovou logikou. Na tento fakt jsem při implementaci narážel opakovaně a z oddělenosti UI a business funkcí jsem těžil. Třetím důvodem je, že pak mohou dva různí lidé s větší mírou nezávislosti pracovat jeden na vývoji UI a druhý na business logice. Nakonec nesmíme opomenout ani to, že automatické testování uživatelských rozhraní je složitější a podléhá jiným zákonitostem než testování doménové logiky.

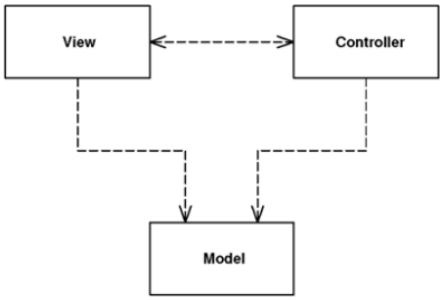
5.1.2 Architektura nájemního portálu

Všechny rozšiřující funkcionality, které jsem v rámci této práce implementoval, byly zapojeny do stávajícího kódu, žádná nestála samostatně. Avšak míra propojení se stávajícím kódem se různila. Funkcionality související s evidencí individuálních studentů jsou součástí již existujících portálů (administrativního a lektorského), takže jejich kód poměrně často interaguje se stávajícím kódem. Oproti tomu funkcionality týkající se pronájmů jsou obsahem úplně nového nájemního portálu, takže při jejich implementaci jsem měl větší míru svobody.

Koncipoval jsem jej jako samostatný portál po vzoru portálu administrativního, lektorského a studentského. Z toho vyplývá, že nájemní portál má svou stránku index.php, což znamená, že zde nejsme svazováni podmínkou použití Render services a můžeme implementovat nějaké čistší a elegantnější řešení svobodněji, než by tomu bylo v případě hlubšího zanoření do stávajícího kódu. Rozhodl jsem se této nově nabyté svobody využít plně a aplikovat v tomto portálu architekturu MVC.

Jak již název napovídá (Model View Controller, struktura je znázorňena na obrázku 5.2), přináší tato architektura dvě důležitá rozdělení. Jednak je to již zmíněné oddělení prezentační logiky, a jednak je to oddělení business logiky a zpracovávání uživatelského vstupu. Obě tato rozlišení v stávajícím kódu naléhavě chybí a pokud budou v budoucnu implementována, výrazně stoupne jeho udržitelnost a čitelnost. Model má v této architektuře na starost business logiku a nevolá ani View ani Controller. View prezentuje obsah klientovi - právě zde by měl být soutředěn html kód a pokud je zde php, pak jen poskrovnu a pouze jednoduchým způsobem: if-statementy, for-cykly, případně gettery. V Controlleru by mělo být obsaženo co nejméně business logiky. Controller zpracovává http požadavky, na jejich základě volá model, od nějž získá potřebná data, která předá view, a ta vrátí klientovi. V praxi se setkáváme se situací, kdy je controllerů více, například, když chceme seskupit podobné typy požadavků do různých tříd, abychom se vyhnuli příliš

velké, jediné třídě Controller, která by vykonávala mnoho vzájemně příliš nesouvisejících funkcí. To je i můj případ. Martin Fowler tyto dílčí controllery označuje pojmem Page Controller, což je však, jak podotýká Reiersol [4, p. 434], poněkud nejednoznačné, protože ve webové aplikaci nezacházíme jen se statickými stránkami a to, oč uživatel žádá nebo co se mu vrátí jako odpověď je složitější a zcela jistě to nelze vždy označit za „stránku.“



Obrázek 5.2: UML reprezentace MVC podle M. Fowlera [6]

Pro MVC jsem se rozhodl proto, že přináší i další výhody kromě oddělení prezentační/doménové/databázové logiky. Podporuje vyšší reuse Views a komponent (jako je v našem případě třeba komponenta Pager), dokáže velmi přehledně zpracovávat asynchronní volání, což je důležité. Dříve totiž měla všechna asynchronní volání na starosti třída Ajax.php. Tato třída je již nyní obrovská a nepřehledná, protože obsahuje mnoho funkcí, které spolu nijak nesouvisí, a to je antipattern. Zavedení architektury MVC umožní ji prozatím alespoň nerozšiřovat a při případném refactoringu někdy v budoucnosti ji zredukovat nebo úplně odstranit, protože jednotlivá volání bude možné seskupit do odpovídajících controllerů. To bude logičtější a přehlednější. Architektura MVC dále také umožňuje optimalizaci pro vyhledávací enginy a usnadňuje formátování dat vrácených modelem i jinak než jako HTML dokument, což ale v této práci využito není.

Je zjevné, že použitím MVC v nájemním portálu vzniká návrhový problém. Tím je dichotomie mezi touto částí aplikace (MVC) a již dříve naimplementovanými portály (většinou nic, ale místy některé aspekty Layers). Kdybych tuto aplikaci vyvíjel od začátku, aplikoval bych právě MVC, protože mi pro tento typ aplikace připadá logické. Pro stávající funkcionality ale použít nešlo, aniž bych významně narušil (nebo neúměrně zdlouhavě a pracně změnil) strukturu stávajícího kódu. Na druhou stranu je nájemnický portál relativně self-contained, s ostatním kódem je propojen téměř výhradně přes model, který často pouze převolává již existující logiku v services, takže je i možné (i když ne nutně žádoucí), aby se zbytek systému dále vyvíjel bez ohledu na MVC použité v tomto portálu. Domnívám se, že řešení, které dává největší smysl, je refaktorovat stávající portály, aplikovat MVC i tam a již existující logiku v mnou implementovaných metodách přesunout do odpovídajících modelů a controllerů.

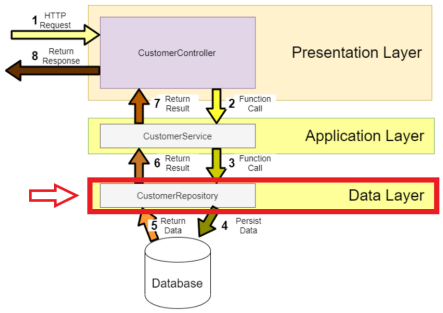
Kapitola 6

Implementace rozšiřujících funkcionalit

V této sekci se budu blíže věnovat některým technickým aspektům implementovaných funkcionalit. Postupovat budu podle vrstev vrstevnaté architektury, a sice od spoda, začnu tedy perzistentní vrstvou. Cílem této kapitoly je také ukázat, že mé řešení implementace nových funkcionalit je přehlednější než stávající kód.

6.1 Perzistence a DAO vrstva

Perzistence je schopnost dat přežít restart aplikace. Metod, jak jí lze dosáhnout je vícero, pro tuto práci je však relevantní jejich ukládání do databáze (ať už relační, key-value store, object store, nebo jiné). Následující obrázek znázorňuje, ve kterém segmentu vrstevnaté architektury se nyní pohybujeme.



Obrázek 6.1: Perzistenční vrstva ve vrstevnaté architektuře [6]

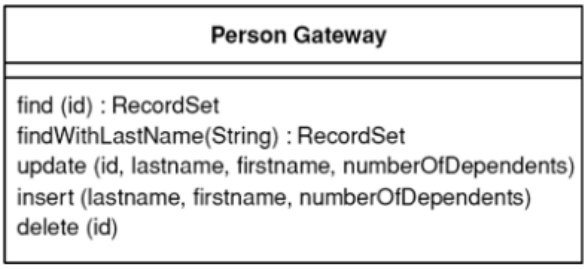
Lingua Nostra pro perzistenci svých dat používá MySQL, což je open-source databázový server. Jedná se o jednoduchou a uživatelsky přívětivou alternativu ke komerčním řešením typu Oracle Database, SQL Server od Microsoftu nebo D2B Universal Database od IBM. MySQL má řadu výhod, jako je vysoká rychlost - na procesoru Pentium Quad Core 7000 bylo schopno dosáhnout 13 000 dotazů za sekundu (údaj z roku 2018, viz [3]). Původní tvůrce Mysql Monty Widenius jej údajně vytvořil právě proto, že byl nespokojen s rychlostí komerčně dostupných databází v jeho době. Mezi další výhody patří vysoká spolehlivost, nízké požadavky na hardware - je schopno běžet i jen s 4MB

operační paměti (tamtéž), dobrá škálovatelnost, multiplatformnost a podpora ODBC.

Pro přístup k databázi je používáno rozhraní MySqli. Toto rozhraní je oproti např. PHP PDO, které je také široce používané, použitelné pouze s MySQL. Jedná se o náhradu za starší mysql, které mělo mnoho nedostatků a ve verzi 7 bylo z PHP zcela odebráno. Oproti PDO má ale výhody v podobě možnosti provádět asynchronní požadavky, provádět více dotazů najednou, získat víc informací o řádcích ovlivněných provedeným dotazem nebo například metody pro uzavření spojení. Obě rozhraní umožňují využívat prepared statements, které ve své implementaci DAO vrstvy využívám i já.

Prepared statements umožňují zvýšit efektivitu při opakovaném provádění jednoho SQL dotazu. Ten je nejprve připraven (v implementaci mysqli je to metoda prepare()), což znamená, že sql výraz projde procesem parsování, zkompiluje se, je na něm provedena optimalizace a je uložen pro pozdější použití. To nastane, když aplikace naváže do připraveného statementu konkrétní hodnoty parametrů pomocí funkce bind(). Následně je výraz funkcí execute() proveden. Hlavními výhodami prepared statements jsou tedy snížení času spotřebovaného na parsování, protože to proběhne pouze jednou, i když je dotaz vykonán víckrát, nižší zatížení sítě, protože se pokaždé posílají pouze parametry a ne celý statement. Další důležitá výhoda je zvýšení bezpečnosti aplikace díky ochraně před SQL injection.

Z hlediska aplikační logiky je práce s mysqli a prepared statements zaobalena do návrhového vzoru vzor DAO - Data Access Object. Martin Fowler jej označuje pojmem Table Data Gateway [6, p. 141], protože tento název vystihuje, že v jeho pojetí se jedná o konkrétní aplikace obecného návrhového vzoru Gateway a také kvůli možnému konfliktu se specifickým významem pojmu Data Access Object ve světě Microsoftu. Já používám název DAO, protože jsem si jej osvojil v průběhu studia. Tento návrhový vzor je znázorněn na následujícím obrázku.



Obrázek 6.2: Příklad návrhového vzoru Data Table Gateway podle M. Fowlera [6]

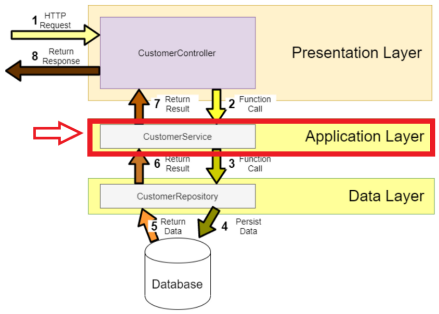
Smyslem tohoto návrhového vzoru je abstrahovat od SQL syntaxe a také obecně poskytnout rozhraní pro práci s databází. Je toho dosaženo tím, že DAO vystavuje jako své rozhraní metody poskytující CRUD operace (nebo i další) s danou entitou - typicky implementujeme separátní DAO pro každou modelovou entitu. Pro lepší čitelnost kódu by bylo vhodné použít generický typ, to však PHP nepodporuje. Alespoň jsem tedy vytvořil základní třídu

Base Dao, která je předkem všech ostatních Dao.

Base Dao výrazně snižuje duplicity v kódu způsobené použitím prepared statements - neustálé opakování prepare, bind a execute lišících se pouze v parametrech a neustálé kontrolování všech těchto metod, zda nevrátily chybový kód. Funkce bind se jevila jako problematická pro refactoring, protože byla volána pokaždé s jiným počtem parametrů, zde se ale naskytlo elegantní řešení pomocí operátoru „...“ (spread), který umožňuje rozložit pole na seznam parametrů. U těch modelových entit, které jsou implementovány jako „dto“ se mi podařilo abstrahovat metodu pro jejich instanciaci v DAO, protože jim jako parametr stačí předat řádek sql výsledku. Naopak u těch modelových entit, které využívají klasické private atributy a getter/setter metody jsem na žádnou elegantní abstrakci nepřišel, neboť jako atributy konstruktoru je třeba předat hodnoty na jednotlivých klíčích výsledku sql dotazu.

6.2 Vrstva doménové logiky a services

Nyní se posouváme o jednu vrstvu výše, a sice k doménové logice. Jako doménovou logiku označujeme tu část aplikace, kde se pracuje s daty, probíhají výpočty atd. Následující obrázek ukazuje, v jakém segmentu třívrstvé architektury s právě nacházíme.



Obrázek 6.3: Vrstva doménové logiky ve vrstevnaté architektuře [6]

Jak již bylo zmíněno v sekci Architektura, aplikace typu stávajícího systému Lingua Nostra příliš mnoho logiky neobsahují. Hlavním úkolem je prezentování dat uživateli a provádění CRUD operací na nich. V důsledku toho spočívá činnost většiny services pouze ve volání DAO vrstvy a předávání získaných dat zpět controlleru.

Zároveň však nelze tvrdit, že by v services neprobíhala vůbec žádná doménová logika. Můžeme uvést například backend validaci odeslaných formulářů, zejména formulářů tvorby nových individuálních studentů, nájemníků a formulářů jejich následné editace. Validace sice probíhá už na front endu, na tu ale nelze z podstaty věci spoléhat.

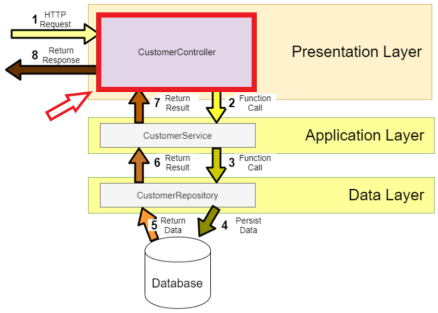
Dále zde v mnoha případech probíhá manipulace s časovými záznamy. To je důležité při bookování slotů dopředu, neboť zde je potřeba najít případné kolize s již zabookovanými sloty. Tudíž je v mnoha případech potřeba přičítat

k hodinám a dnům týdny. S tím souvisí i výpočet data konce období z jeho začátku, který je uložen v databázi. Do této skupiny funkcí patří i vybírání slotů pro bookování mezi dvěma vybranými daty (je potřeba odfiltrovat sloty vyplněné výukou a již existujícím bookingem). Také sem můžeme zařadit některé elementární transformace dat, například převod pole slotů na asociativní pole s názvem místnosti a českým jménem dne (aby tuto logiku nemusely vykonávat komponenty).

Z komplikovanějších funkcionalit se sem řadí zejména posílání faktur do Fakturoidu a zachycování webhooků. Je třeba rozlišovat mezi fakturami na firmu a fakturami vystavované osobám bez IČO. Dále je také třeba správně vyhodnocovat odpovědi, které z Fakturoidu přicházejí. Významnou důležitost má i generování e-mailových upozornění mj. v případě, kdy administrátor zruší slot nájemníkovi.

6.3 Render Services a MVC Controller

Následující dvě subsekce se zabývají nejvyšší vrstvou třívrstvé architektury, a sice tou prezentační. Zde se soustředím na controllery, tedy na tu část systému, která zachycuje příchozí požadavky, dál rozhoduje o jejich zpracování a následně vrací odpověď. Následující obrázek ukazuje, kde se právě nacházíme:



Obrázek 6.4: Controllery v prezentační vrstvě vrstevnaté architektury [6]

V této subsekci bych se chtěl vypořádat s nejasnostmi ohledně snad nejméně přehlednou částí celého řešení, a sice s typem tříd, které jsem označuji jako Render Services. Aby bylo zřejmější, proč jsem je takto pojmenoval a k čemu vlastně slouží, je nutné nejprve nastínit strukturu stávajícího kódu, která v aplikaci hraje přibližně roli controlleru.

Jádrem administrativního portálu, což je oblast, kde se soustřeďuje největší množství logiky, je třída AdminService. Navzdory názvu se však o service nejedná, alespoň ne ve smyslu service jako prvku servisní vrstvy ve vrstevnaté architektuře - ta by měla odpovídat nějakému endpointu rozhraní, a provádět doménovou logiku. AdminService se ale chová daleko spíše jako controller. Při dotazu na index.php v administrativním portálu tato třída instancuje nějakou dílčí „Service“ v závislosti na url. Index vrátí část výsledného html (hlavičku, navbar atd.) a pak se zavolá metoda render() dodané služby. Tím je dosaženo toho, že se veškerá aktivita odehrává na index.html.

Příkladem dílčí Service je třeba ApplicationService, která sice opravdu sdružuje logiku týkající se přihlášek, zároveň však mísí doménovou, databázovou i business logiku, a to i v jedné metodě. Ani tu tedy nelze za service ve výše definovaném smyslu slova označit.

Asynchronní volání byla řešena tak, že na začátku index.php se nachází skript, který kontroluje, jestli v poli „action“ v proměnné $_GET příchozího požadavku je nastavena hodnota. Pokud ano, je tato hodnota předána třídě Ajax.php, která slouží vlastně jako univerzální controller pro všechna asynchronní volání. Tato třída je realizována rozsáhlým if-stromem, který vrací html podle hodnoty proměnné „action“, která mu byla předána.

Chtěl-li jsem při implementaci nových funkcionalit dodržet tuto strukturu administrativního portálu, nezbylo mi, než přidat další „Service.“ To však neznamená, že i tyto nové třídy budou fungovat s naprosto stejně neoddělenou logikou. Protože všechny původní „Services“ dědí z AdminService, mají předepsanou metodu render, která se volá vždy, když se jedná o synchronní volání, tzn. není nastavena hodnota getového pole „action“. Rozhodl jsem se nové třídy pojmenovat RenderService, abych dal najevo, že vykonávají metodu render() a že neodpovídají services ve smyslu Layered architektury.

RenderServices jsem pojal jako jakési „sub-controllery“ a vyčlenil jsem z nich databázovou a prezentační logiku. Databázovou logiku nově vykonává DAO vrstva (viz výše) a prezentační logiku jsem přesunul do php tříd, které označuji pojmem „component.'"K problematice komponent viz subsekce Prezentační vrstva. Doménovou logiku jsem přemístil do nově vzniklých tříd v adresáři Services, které se již skutečně jako services chovají, tzn. vykonávají pouze doménovou logiku.

Bylo zřejmé, že i ze sekcí portálu reprezentovaných novými RenderServices budou chodit asynchronní volání. Nechtěl jsem však rozšiřovat již tak příliš rozsáhlé Ajax.php, a proto jsem se rozhodl přidat nějakou formu endpointů pro vzdálená volání do RenderServices. V zájmu zachování jednotnosti a předvídatelnosti jsem zachoval mechanismus rozhodování podle pole „action,“ ale kód pro zpracování nových asynchronních volání jsem přesunul do RenderServices a na index.php jsem volání Ajax.php předřadil podmínku, která nechá asynchronní volání zpracovat RenderService, pokud pro něj obsahuje endpoint, a teprve pokud ne, zavolá Ajax. Domnívám se, že tento způsob umožní do budoucna nerozšiřovat a při případném refactoringu třeba i postupně zmenšit až odbourat třídu Ajax, jednotlivá volání seskupit do odpovídajících Services a přeměnit je v plnohodnotné controllery a časem třeba i v RESTful rozhraní.

Lektorský portál je řízen namnoze logikou if-then-else a jeho refaktoring nebyl obsahem zadání. Provedl jsem v něm jen minimální změny. Jmenovitě je to přidání kalendáře a úprava lektorského výkazu, aby zahrnoval všechny položky potřebné pro rozšířenou evidenci individuálních studentů, tzn. dropdown, který načítá individuální studenty z databáze, date picker pro datum lekce, radio button group pro délku lekce a automaticky vyplněná pole s cenou.

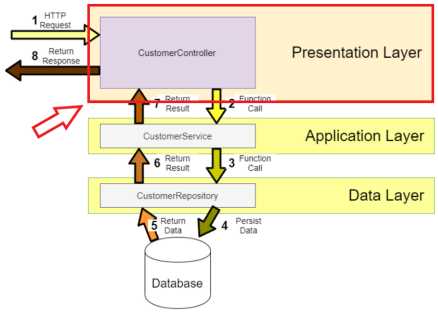
V nájemním portálu je controllerová logika přehlednější. Index.php nájemního portálu pouze instancuje router (třída Bootstrap.php) a předá mu

řízení. Ten využívá toho, že PHP umožňuje instancovat třídu pomocí new a připojení závorek k jejímu jménu předanému proměnnou a přímo transformuje odpovídající část url požadavku na konkrétní controller, který je potřeba instancovat. Stejným mechanismem umožňuje zavolat specifickou metodu daného controlleru, přičemž není-li žádná metoda specifikována, volá se metoda index.

Tyto specifikované metody plní většinou roli endpointů pro asynchronní volání a vrací pouze views bez layoutu (blíže viz následující subsekce.) Metoda index oproti tomu funguje jako homepage jednotlivých stránek a vrací views v zadaném layoutu. Controllery jsou rozděleny na více různých kvůli seskupení souvisejících funkcí a větší přehlednosti. Oproti RenderServices a hlavně původním Services se zde mnohem lépe daří udržet pouze controllerovou logiku, tzn. získávání informací z modelu, jejich předávání views a vracení těchto views jako odpovědi.

6.4 Prezentační vrstva

Nejvyšší vrstvou třívrstvé architektury je vrstva prezentační. Kromě controllerů by zde neměla být prováděna téměř žádná aplikační logika (s výjimkou if-statementů a jednoduchých cyklů) a naopak by se zde měl soustředit pokud možno veškerý html kód. Je-li toto dodrženo, je kód aplikace mnohem přehlednější a jeho vývoj udržitelnější. Pozici této vrstvy v třívrstvé architektuře znázorňuje následující obrázek:



Obrázek 6.5: Prezentační vrstva vrstevnaté architektury [6]

Domnívám se, že oddělení prezentační logiky od ostatních oblastí, byť nedokonalé, je pro aplikaci důležitým přínosem. V administrativním portálu reprezentuje prezentační logiku třídy, které označuji jako komponenty. Při návrhu části aplikace, která se zabývala individuálními studenty a kalendářem jsem se pod vlivem struktury dosavadního kódu rozhodl pojmout jednotlivé komponenty jako php třídy s metodou render(), která vrací výsledné html. Toto vede k méně zřejmému a méně ostrému oddělení prezentační logiky, protože to svádí k dojmu, že protože se jedná o třídy, tak je možné, aby měly nějaké chování, tzn. do nich vkládat logiku, což jsem i já v několika případech udělal. Například jsem použil návrhový vzor Strategy pro vykreslování slotů v kalendáři, což umožnilo jeho přepoužití v administrativním i v lektorském

portálu. Jediným rozdílem je zde totiž to, jak se komu který typ slotu zobrazí (zda s informacemi či bez nich, zrušitelný/ nezrušitelný apod.)

Nicméně na obhajobu tohoto principu lze uvést, že má i svou výhodu. Views používaná v klientském portálu jsou používaná pomocí include, což znamená, že se v každém případě jejich volání musí nastavit lokální proměnné (navíc dynamicky). Oproti tomu instanciace php třídy jí předá potřebné informace v konstruktoru, je tedy snazší je používat na různých místech v kódu a tok informací je v jejich případě čitelnější. Nicméně hlavní nevýhodou těchto komponentových tříd je nepraktičnost jejich zanořování, protože vede k nutnosti tvořit třídy typu „wrapper," „section" a podobně.

V nájemním portálu používám již standartní php soubory, kde je aplikační logiky minimum. Rozlišuji zde layouty a views. Rozdíl je ten, že samotná views vrací controller typicky v odpovědi na asynchronní volání, jde většinou o aktualizaci nějaké části stránky poté, co byla provedena nějaká změna na datech. Layout spolu s nějakým view se pak použije pro načtení celé stránky. Smyslem používání layoutů je, že nemusíme neustále dokola includovat hlavičky, záhlaví, navbar, zápatí a podobně. Další výhodou pro případný budoucí vývoj je zvýšená flexibilita díky možnosti vykreslování stejného view s jiným layoutem.

Nejproblematičtější na tomto řešení jsou vzniklé duplicity. Konkrétně se jedná o komponenty, které jsou realizovány jak třídou v administrativním portálu, tak php souborem v portálu nájemním. Z toho vyplývá důležitý úkol do budoucna, totiž tyto reprezentace sjednotit. Já osobně se po nabytých zkušenostech přikláním spíše ke standardnějším views a layoutům.

V této sekci jsem stručně popsal technickou stránku implementace a vysvětlil, proč jsou nové funkcionality přehlednější než kód stávajícího systému. Veškerá logika přístupu k datům v databázi se soustředí do perzistenční vrstvy, která používá návrhový vzor DAO a přistupuje k MySQL databázi pomocí mysqli. Doménová logika se přesunula do business vrstvy, probíhá zde např. backend validace. V zájmu kompromisu se strukturou stávajícího kódu slouží jako vstupní body nových funkcionalit RenderServices, v nichž byla ponechána pouze logika odchytávání požadavků, volání business vrstvy a vracení html kódu. Tento jsem přesunul do komponent a zpřístupnil jej metodou render(). Protože je lépe oddělena databázová, doménová a prezentační logika, je nový kód přehlednější a udržitelnější než stávající.

6.5 Některé bezpečnostní aspekty rozšiřujících funkcionalit

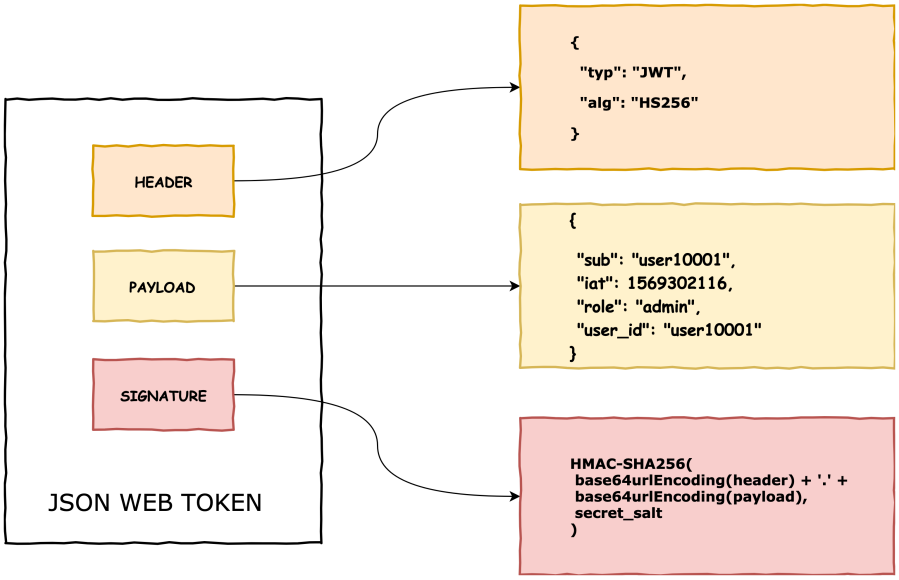
Než téma implementace zcela opustíme a přesuneme se k testování, chtěl bych dát něco prostoru jedné stránce implementace, kterou jsme dosud opomíjeli, ač je velmi důležitá. Touto stránkou je bezpečnost. Podobně jako u testování se i zde jedná o širokou oblast, které by bylo možné věnovat samostatnou rozsáhlou práci. Cílem této práce však není zlepšit zabezpečení stávajícího systému. Budu se tedy věnovat jen některým základním bezpečnostním

prvkům, nejprve udržování session, pak správě hesel a nakonec validaci dat.

Protokol HTTP(s) je, z definice bezestavový, což má sice mnohé výhody, ale v řadě aplikací to představuje problém, protože s nimi uživatelé pracují způsobem, který nějakou mír udržování stavu vyžaduje. Nejinak je tomu i v našem systému. Hlavním problémem je přihlašování uživatelů, abychom oddělili, jaký uživatel má přístup k jakým datům. Způsob, jakým je řešeno přihlašování a udržování stavu v administrativním, lektorském a studentském portálu ponechám stranou a přiblížím řešení implementované v portálu nájemním.

Tradičním způsobem, jímž bylo v php aplikacích řešeno přihlašování uživatelů, byla proměnná session. Pro programátora je velmi praktická, má ale některé vážné problémy: data jsou na serveru uložena v plaintextu, přistupování k těmto datům se děje pomocí čtení souborů (protože mechanismus proměnné session je založen právě na zapisování do souboru), což má za následek race-condition, které php sice umožňuje řešit pomocí session-lock, což způsobí, že proměnná session umožní pouze synchronní přístup. To sice zabrání vzniku nekonzistentních stavů, zároveň to ale z proměnné session udělá bottleneck, který může aplikaci výrazně zpomalit. Dalším problémem je bezpečnost, na session totiž existuje řada útoků, například session fixation (útočník si nechá přiřadit session ID nějakým způsobem přiměje oběť nastavit si session se stejným ID, a pak k ní může přistupovat), session sidejacking (v zásadě man in the middle) a cross-site scripting. Místo session v této práci používám pro udržení stavu a autentikaci přihlášeného uživatele Json Web Token (JWT). Mezi jejich hlavní výhody patří, že oproti např. API Keys obsahují informace a metadata, která mohou sloužit třeba k ověření platnosti tokenu v čase nebo na doméně nebo identitu uživatele. JWTs dále nevyžadují centralizovanou certifikační autoritu a jsou kompatibilní s OAUTH2.

Struktura JWT (viz obrázek 6.6) se skládá z hlavičky, která je zakódovaná pomocí Base64 (kódovací schéma pro reprezentaci binárních dat) a vyjadřuje, jaký kryptografický algoritmus byl použit pro podepsání - umožňuje jak symetrické tak asymetrické algoritmy a typ tokenu, který je vždy nastaven na JWT. Samotnný payload JWT tokenu je také zakódován Base64 a skládá se z takzvaných claimů (tvrzení), která slouží k ověření platnosti tokenu. Uživatel může definovat vlastní, často se ale používají předdefinované (registered) claims, především: iat - timestamp v době vydání tokenu, iss - string vyjadřující integritu vydavatele, nbf - timestamp začátku platnosti tokenu a exp - timestamp konce platnosti. Podpis tokenu se skládá z hlavičky, payloadu a secret value. Pro zvýšení bezpečnosti mají tokeny nastavenou relativně krátkou dobu platnosti.



Obrázek 6.6: Struktura JWT Tokenu

S autentikací souvisí také správa hesel. Triviální řešení, ukládání hesel v plaintextu, je nebezpečné, mimo jiné proto, že pokud by útočník dostal byť jen čtecí přístup k dazabázi, získal by i uživatelská hesla. Také platí, že uživatelé často používají stejná nebo podobná hesla pro více služeb a při získání např. data dumpu naší databáze by došlo k ještě rozsáhlejší kompromitaci.

Lepší řešení je neukládat samotná hesla, ale pouze hashe z nich. V případech výše uvedených útoků by pak útočník získal pouze hashe, z nichž hesla není možné zpětně získat. Kryptografická hashovací funkce totiž musí splňovat hlavně požadavek jednocestnosti, to znamená, že pro obraz Y není možné nalézt vzor X takový, že f(X)=Y, nebo alespoň ne s vynaložením takových zdrojů, které by nepřevýšily hodnotu získané informace. V této práci používám pro implementaci hashování hesel framework PHPass. Oproti nativnímu password_hash a password_verify jej používáme kvůli podpoře starší verze PHP. Hashovací funkce, kterou PHPass používá, je bcrypt. Tato funkce je založená na šifře Blowfish, což je nepatentovaná symetrická bloková šifra Feistelova typu, u níž není dosud známa efektivní metoda jejího prolomení. Podporuje klíče v délce od 32 do 448 bitů, má 16 kol a jejím zvláštním rysem je využití velkých S-boxů, které jsou zde závislé na klíči. V dnešní době má již modernější nástupce Twofish a Threefish. Přesněji řečeno, bcrypt nepoužívá původní Blowfish, ale jeho pozdější modifikovanou verzi Eksblowfish.

Bcrypt používá salting, takže je odolný proti rainbow tables (v zásadě předpočítané tabulky výsledků hashovacích funkcí). Bcrypt je také adaptivní, což znamená, že můžeme zvýšit počet iterací a, tím ji uměle zpomalit. To by se mohlo zdát jako nevýhoda, ale protože toto zpomalení se projeví jen při přihlašování a registraci uživatelů, není tak výrazné. Naopak pomalejší hashovací funkce nám poskytuje mnohem lepší ochranu před bruteforce útokem.

Dalším důležitým aspektem bezpečnosti je validace dat. Jedna vrstva validace uživatelských vstupů sice probíhá na frontendu, ale obecně platí, že na frontendovou validaci nelze spoléhat, protože javascript lze vypnout, požadavky je možné uměle vytvářet pomocí curl a podobně.

Nejkomplexnější část validace je kontrola formulářů pro přidání individuálního studenta a nájemníka a dále editace jejich profilových dat. Validaci těchto vstupů jsem implementoval objektovým způsobem, a sice podle schématu uvedeného v PHP in Action [4, p. 393], jehož některé části jsem modifikoval podle potřeb naší aplikace.

Základní princip je velmi prostý: jednotlivé části vstupu se jedna po druhé validují každá jinou booleovskou funkcí, a pokud jí nevyhovují, přidá se odpovídající chybová hláška do seznamu chyb. Následně pokud je tento seznam neprázdný, formulář validací neprošel a je vrácen tento seznam. Toto by bylo možné implementovat i naivní verzí, kde by se validační funkce skládala z mnoha if-then-else, kde by každé testovalo jednu položku vstupu. Výsledný kód by však nebyl příliš vizuálně atraktivní.

Řešení z PHP in Action se skládá z několika tříd, jejichž princip nemusí být na první pohled zcela zřejmý. Důvodem je to, že se s přijatými daty zachází jako s návrhovým vzorem Command, což je však pro potřeby této aplikacie příliš komplexní, alespoň v tomto stádiu. V jádru celého řešení stojí třída ValidationFacade, která samotnou validaci provádí a poskytuje mj. metodu isValid().

Hlavní výhoda toho řešení spočívá v jeho modularitě a přepoužitelnosti. Té je dosaženo způsobem, jakým funguje metoda validate() ve ValidationFacade: vytvořené instanci se nejprve přiřadí jednotlivé validátory podle toho, jaká pole chceme validovat, jakým způsobem a s jakou chybovou hláškou v případě neúspěchu, a když je zavoláno validate(), pak ValidationFacade volá jeden po druhém své validátory na přijatá data.

Třída validátor není mimochodem úplně triviální. Sama totiž validační funkci neprovádí; pouze rozhoduje o tom, jestli validace pole uspěla a ukládá chybovou hlášku do ValidationFacade. Dále agreguje třídy typu Specification (design pattern Specification), které již umí validovat specifikované pole.

ValidationFacade tedy můžeme konfigurovat podle aktuálních potřeb. V případě validace individuála přidáme validátory na cenu odučených lekcí a metodu platby, které u nájemníka potřeba nejsou. V případě editace profilu zase nebudeme chtít validovat e-mail, protože ten editovat nelze (editace e-mailu není žádoucí z hlediska integrity dat). Po skončení validace můžeme na ValidationFacade zavolat metodu getErrors(), která nám předá nasbírané chybové hlášky. S nimi pak můžeme dále zacházet, jak uznáme za vhodné.

6.6 Testování

Testování softwaru je rozsáhlá a komplexní oblast, která je ve vývoji nezastupitelná. Už jen samotné testování i relativně malého informačního systému by svým rozsahem mohlo vydat na samostatnou bakalářskou práci nebo i mnohem víc. V prostředí tohoto reálného projektu jsme také výrazně limi-

továni zdroji, nemáme speciální testovací tým - na straně klienta si musíme vystačit s lektory, administrativními pracovníky a jedním vývojářem, jejichž možnosti jsou také časově limitované. Proto jsem se v testovací části projektu omezil na demonstraci základních principů a ilustrativní příklady, spíše než na testování v tak komplexní podobě, jak je prezentuje např. Software Testing [9], v takovém rozsahu by ani nedávalo smysl. Dvě hlavní oblasti, na něž se zaměřím, jsou jednotkové testování a akceptační testy.

6.6.1 Jednotkové testování

Jednotkové testování spočívá v tom, že píšeme kód, který testuje kód. Je postaveno na myšlence, že kvalita by měla být zabudována do samotného procesu tvorby softwaru. [4, p. 191] Toto je také jedna z centrálních myšlenek TDD (Test-driven development), což je technika vývoje, kdy píšeme nejdříve testy, pomocí nichž specifikujeme, jak by měla testovaná jednotka fungovat a pak postupně refaktorujeme, aby testy procházely na reálném kódu. Původně v naší aplikaci nebyly unit testy vůbec žádné, testování se spoléhalo vlastně jedině na znalost fungování organizace a uživatelské testy. Součástí úkolu tedy bylo také nastavit testovací framework a založit strukturu testovacích adresářů. Toto je kromě testů samotných také dalším (dle mého názoru ne zcela nepodstatným) vedlejším přínosem.

V této práci používám pro jednotkové testování framework PHPUnit, který není nepodobný např. velmi rozšířenému JUnitu v Javě. Na rozdíl od něj však PHPUnit definuje Test Suites pomocí xml a nikoli jako javovskou třídu, čímž připomíná spíše TestNG. Při spuštění načte PHPUnit ze své konfigurace všechny test suites a pro každou vykoná všechny testovací metody ve třídách obsažených v adresáři, který je uvedený u dané test suite.

Aby byl unit test unit testem, neměl by záviset na kódu třetích stran. Měl by testovat jednu velmi úzce vymezenou funkcionalitu přehledným způsobem. V některých případech se však nějakému propojení s kódem mimo testovanou metodu nevyhneme. V takových situacích je vhodné použít techniku mockování. Myšlenka je taková, že místo abychom doopravdy instancovali nějakou třídu, instancujeme jen její napodobeninu, které předepíšeme, jaké konkrétní výsledky má vracet při volání svých metod. Docílíme tak toho, že chyby, které odhalíme, budou skutečně chybami v testované metodě a ne v nějaké jiné části kódu, kterou tento test přímo netestuje. Zde používám Mockery, což je jednoduchý a flexibilní framework, který poskytuje snadnou integraci s výše uvedeným PHPUnitem.

Jak již bylo řečeno, tato aplikace má své těžiště v prezentaci dat z databáze grafickým způsobem uživateli. SQL dotazy a renderování HTML nejsou oblasti, kde by se síla unit testů demonstrovala nejjasněji. Vytvořil jsem jen několik jednoduchých testů, které se zabývají hlavně těmi oblastmi aplikační logiky, jako je práce s datem a časem a jejich formátování nebo převod polí objektů indexovaných numericky na asociativní pole s identifikátorem jako klíčem a podobně.

Obecně platí, že hlavní výhodou unit testů je to, že poskytují účinnou ochranu proti zavlečení chyb (testy, které dřív procházely, po implementaci

nové funkcionality procházet přestanou - vidíme to hned), jejich snadné a rychlé spuštění a také to, že rychle odhalí ty nejvážnější chyby způsobující pád aplikace, pak s nimi nezdržujeme testery. Zároveň je však potřeba mít na paměti, že vysoký počet unit testů nemusí automaticky znamenat dobré pokrytí aplikace. Unit testy také v žádném případě nejsou dostatečnou náhradou za uživatelské testování.

6.6.2 Uživatelské akceptační testy

Informační systém se skládá z mnohem více prvků, než pouze z kódu. Jeho součástí jsou také data, procesy a v neposlední řadě lidé, kteří jej používají. Testovat pouze kód je příliš málo. Může se velmi reálně stát, že kód bude v pořádku, a přesto informační systém jako celek nebude fungovat tak, jak se očekává, protože chyba bude ležet úplně jinde. Pomyslnou branou, která by měla propustit co nejlepší software, jsou uživatelské akceptační testy - UAT [2, p. 17].

Při UAT vstupuje do testování výrazně hledisko uživatele. UAT testují víc než jen funkčnost, výkon, spolehlivost nebo bezpečnost. Mají hlavně ukázat, zda systém doopravdy poskytuje očekávané funkcionality, když s ním zachází opravdový uživatel. UAT na tuto otázku dokáží odpovědět, protože testují produkt proti samotnému účelu, za nímž byl navržen. Důležitější než strukturovanost je práce s reálnými daty a realistickými scénáři.

Aby byly UAT efektivní, musíme stanovit, čeho mají dosáhnout - kdy můžeme přestat testovat a pustit systém do provozu. Naivní odpověď by mohla znít: až bude systém fungovat korektně, nebude mít žádné defekty a bude připraven na spuštění v datum release day. To však není reálně možné. Čelíme typickému multikriteriálnímu problému a nejspíš dokážeme nalézt řešení přinejlepším paretovsky optimální. Přistoupíme tedy na určitý kompromis - tím budou naše akceptační kriteria.

Rozdělme tedy funkcionality na tři skupiny: zásadní funkce, bez nichž nebude podnik moci poskytovat své služby; důležité funkce, které nejsou kritické, ale systém bez nich nebude fungovat tak, jak se očekávalo (např. budou existovat workaroundy, ale ty budou časově náročné nebo budou vyžadovat další zaškolení zaměstnanců), a funkce kosmetické, které zvyšují použitelnost nebo šetří čas uživatelů. Analogicky můžeme rozdělit i chyby. [2, p. 144].

Spolu se zástupci Lingua Nostra jsme došli k rozhodnutí, že při tvorbě akceptačních kriterií budeme vycházet z požadavku, aby se u zásadních funkcí nevyskytovaly kritické a vážné chyby a u důležitých funkcí aby nedocházelo k kritickým chybám. Konkrétní transformace těchto pravidel na implementované funkcionality nám dala seznam akceptačních kriterií. Při jejich formulaci jsem vycházel z user-stories driven AC, viz [1]) Seznam hlavních kriterií je obsažen v příloze.

6.6.3 Testovací strategie

Nyní, když máme definovaná akceptační kriteria, byl by další logický krok použít své znalosti testování, abych určil, jak tato kritéria praktickým a efektivním způsobem splnit, tzn. stanovit testovací strategii, abychom definovali co a do jaké míry je třeba otestovat a na základě této strategie vybudovat testovací plán.

Toto je postup, který navrhují Hambling a van Goethem [2, p. 105]. My však musíme mít na paměti, v jakém prostředí se nacházíme - jde o malý projekt pro malý podnik a v takové situaci je tvorba detailní strategie a podrobného plánu pro naše potřeby příliš rozsáhlá. Přestože by bylo možné použít akceptační kriteria přímo pro definování samotných testů, bude vhodné nějaký přiměřeně detailní testovací plán přeci jen vytvořit.

Základem testovací strategie je popis systému. Ten musí obsahovat test goals - máme user-story driven akceptační kriteria, proto je vhodné (a rozsahově přiměřené) je rovnou použít jako test goals. Další potřebnou informací jsou procesy, jichž se test goals týkají: ty převezmeme v podobě use-case scenarios ze semestrálního projektu. Obdobně převezmeme i v semestrálním projektu definované funkční požadavky. Zbývá rozdělení systému na moduly. Domnívám se, že je účelné převzít členění, které Lingua Nostra běžně používá a je ostatně obsaženo i v semestrálním projektu, tedy rozdělení na administrativní, lektorský a nájemní portál. Popis systému nám umožní si ujasnit, zda jsou test goals pokryty funkčními požadavky, zda pro každou funkcionalitu je funkční požadavek a vice versa. Popis nejdůležitějších částí systému tedy vypadá následovně:

Test Goal	Proces	Požadavek	Modul
I1	Admin přidá individuála	FR 4.1	Admin
I2	Lektor zapíše výuku	FR 4.7	Lektor
I3	Admin fakturuje individuála	FR 4.9	Admin
I4	Admin fakturuje individuála	FR 4.4	Admin
I5	Lektor zapíše výuku	FR 4.6	Admin
N1	Admin přidá nájemníka	FR 3.2	Admin
N2	Nájemník se přihlásí	FR 3.4	Nájem
N3	Nájemník se odhlásí	FR 3.4	Nájem
N4	Nájemník bookuje externě	FR 3.6	Nájem
N5	Nájemník ruší slot	FR 3.8	Nájem
N6	Nájemník bookuje externě	FR 2.3	Nájem
N7	Admin fakturuje nájemníka	FR 2.3.3	Admin
N8	Admin fakturuje nájemníka	FR 3.5	Admin
A1	Admin ruší slot nájemníka	FR 2.1.2	Admin
L1	Lektor bookuje interně	FR 1.2	Lektor
L2	Lektor bookuje interně	FR 1.6	Lektor

Tabulka 6.1: Popis systému

Následujícím krokem je určení testovacích priorit, z nichž bude následně plynout intenzita testování jednotlivých oblastí. Priority je nejlépe odvodit z rizik, která plynou z provozu testované aplikace v rámci organizace. Naivní řešení by bylo přidělit všem funkcionalitám maximální prioritu, protože přeci chceme, aby vše fungovalo perfektně. Toto by ale nikam nevedlo, protože náš projekt je malý a naše zdroje jsou omezené, nedokážeme tudíž stoprocentně pokrýt všechny oblasti. Realistické ohodnocení musí vycházet ze dvou faktorů: jednak z pravděpodobnosti selhání a jednak z možného poškození (výše dopadu, intenzita negativního účinku). Pravděpodobnost selhání budeme určovat pro funkcionality spíše než pro moduly, protože každý modul obsahuje vícero funkcionalit, jejichž pravděpodobnost selhání se významně různí. V úvahu zde budeme brát především zkušenosti z vývoje, frekvenci používání a interakci s rozhraními třetích stran. Možné poškození budeme vyvozovat z funkčních požadavků a při jeho vyhodnocování budeme vyšší hodnoty přiřazovat těm požadavkům, které souvisí s finančními transakcemi a bezpečností informací, menší hodnoty pak těm, jejichž cílem je QoL a podobně.

Proces	Podproces	Req.	Prav. selhání	Míra poš.
1. Admin přidá individuála	validace	FR 4.1	M	L
2. Admin přidá individuála	uložení	FR 4.1	M	H
3. Lektor zapíše výuku	výkaz	FR 4.7	H	H
4. Lektor zapíše výuku	kredit	FR 4.2	L	H
5. Admin fakturuje individuála	tvorba	FR 4.9	H	M
6. Admin fakturuje individuála	placení	FR 4.9	M	M
7. Admin přidá nájemníka	validace	FR 3.3	M	L
8. Admin přidá nájemníka	uložení	FR 3.2	M	H
9. Nájemník se přihlásí	přihlášení	FR 3.4	L	H
10. Nájemník se odhlásí	odhlášení	FR 3.4	L	L
11. Nájemník bookuje externě	základní booking	FR 3.9	L	H
12. Nájemník bookuje externě	booking dopředu	FR 3.9	M	L
13. Nájemník ruší slot	booking zrušení	FR 3.7	M	M
14. Admin fakturuje nájemníka	tvorba	FR 2.3.3	H	H
15. Admin fakturuje nájemníka	placení	FR 3.5	H	H
16. Admin ruší slot nájemníka	placení	FR 2.1.2	H	L
17. Lektor bookuje interně	rozvrh	FR 1.2	M	L
18. Lektor bookuje interně	booking	FR 1.6	M	M

Tabulka 6.2: Pravděpodobnosti selhání a míry poškození

Poslední, co je potřeba udělat, než budeme moci vybrat testovací techniky a vytvořit testovací scénáře, je stanovit třídy rizika. Třídu A budeme testovat nejintenzivněji, třídu B se střední a třídu C s nejnižší intenzitou. Třídy rizika budeme určovat podle kombinace pravděpodobnosti selhání a míry rizika. Vzhledem k omezenosti zdrojů nemůžeme dopustit, aby třída A byla příliš

rozsáhlá. Výrazná většina případu musí spadnout do třídy C nebo B. Pro rozdělení tříd jsem stanovil následující tabulku, kde sloupec reprezenzuje míru dopadu a řádek pravděpodobnost vzniku:

P/M	L	M	H
L	C	C	B
M	C	C	B
H	B	B	A

Tabulka 6.3: Třídy rizika podle pravděpodobnosti selhání (P) a míry poškození (M)

6.6.4 Návrh testů a příklady scénářů

Podstata fungování aplikace příliš nenahrává testování průchodů programem, protože fungování tohoto systému vlastně žádnou lineární oblast nemá, ve většině případů tedy testujeme kombinace vstupů a integritu dat. Obzvlášť integrita dat je pro nás důležitá, protože na základě stavu slotů a odučených lekcí se rozhoduje o využití místností a o finančních transakcích. S ohledem na tabulku tříd rizika jsem navrh konkrétní techniky pro jednotlivé subprocesy. Ty jsou obsaženy v příloze.

Uveďme nyní příklady některých scénářů použitých při testování. U jednotlivých testů vždy uvádím id testu, jeho stručný název, hloubku detailu, shrnutí toho, co test testuje, popis jeho provedení, vstupní podmínky, které musí být splněny před jeho začátkem, testovací data, která poslouží jako vstupy do konkrétního testu, a výsledek, který po provedení testu očekáváme.

Údaje	Obsah
ID testu	1
Název	Validní hodnoty pro Add Indiv Form
Hloubka detailu	Nízká
Shrnutí	Platné hodnoty - happy path přidání individuála
Popis	Zadáme vstupní hodnoty tak, aby vyhovovaly specifikaci
Vstupní podmínky	V db není individuál se stejným e-mailem
Testovací data	Údaje klienta Tobiáš Malý z interní evidence
Očekávaný výsledek	Formulář se odešle na server

Údaje	Obsah
ID testu	7
Název	Životní cyklus individuála
Hloubka detailu	Střední
Shrnutí	Deaktivace a zobrazení profil+seznam
Popis	Zobrazíme v profilu a seznamu, deaktivujeme a opakujeme
Vstupní podmínky	Vytvořený individuál
Testovací data	Zde nepotřebujeme
Očekávaný výsledek	I deaktivovaný individuál bude dál vidět i na profilu

Údaje	Obsah
ID testu	19
Název	Životní cyklus faktury individuála
Hloubka detailu	Střední
Shrnutí	Založit, zaplatit a zkontrolovat kredit
Popis	Ze seznamu, ve fakturoidu dát jako placenou a check kredit
Vstupní podmínky	Vytvořený student
Testovací data	Validní výše faktury
Očekávaný výsledek	Faktura bude vidět jako zaplacená a zvýší se kredit

6.6.5 Vyhodnocení uživatelského testování

Přestože jsme v průběhu uživatelského testování na žádné vážné chyby až na jednu výjimku nenarazili, podařilo se nám brilantně demonstrovat dobře známé nevýhody vodopádového modelu vývoje softwaru. Zjistili jsme, že účel některých prvků aplikace chápeme já jako vývojář a administrativní pracovníci jako uživatelé velmi odlišně. Skvělým příkladem je například to, že uživatelé počítají s tím, že víc individuálních studentů může mít stejnou e-mailovou adresu. V současné fázi implementace už by ale tato změna byla neúměrně náročná, protože e-mail je cizím klíčem, jehož pomocí jsou individuální studenti propojeni s fakturami. Dále jsem se dověděl o existenci kurzů, které netrvají 60 nebo 90 minut. Jsou však pouze dva, je to tedy možné řešit pomocí poznámky k tomuto individuálnímu studentovi. Toto je také typickou nevýhodou vodopádového modelu: mnoho problémů je odhaleno pozdě a je tudíž mnohem nákladnější je opravit, než kdyby byly odhaleny dříve.

Také spontánně vznikly úplně nové požadavky na funkcionality, jako je například abecední řazení individuálních studentů v dropdownu ve výkazu lektorů. Tento požadavek je vlastně velmi logický: individuálních studentů je už v současné době vysoké množství a dále přibývají. Lektoři jsou z velké části Italové a přesné zapamatování si českých jmen jejich klientů je pro ně mnohdy náročné, a proto pro ně bude abecední řazení vítaným ulehčením. Shromažďování funkčních požadavků jsme ve fázi semestrálního projektu věnovali značnou pozornost, přesto se nám na požadavky tohoto typu nepodařilo přijít. Naštěstí dodatečná implementace této konkrétní funkcionality byla banalitou, která zabrala sotva několik minut.

Jedním ze způsobů, jakým lze těmto situacím předejít, by bylo použití jiné metodiky vývoje, například Incremental Delivery. Zde však narážíme na problém, protože tato metodika (jako i některé další agilní metody) vyžaduje velice častou přítomnost zákazníka a jeho účast na vývoji, což při současném stavu fungování Lingua Nostra nebylo dost dobře možné. Obecně však dopadlo uživatelské testování dobře, do značné míry díky tomu, jak kvalitně a předvídavě byl zpracován návrh funkcionalit pro individuální studenty. Chyby byly opraveny a převážná část funkcionalit je již v tuto chvíli nasazena na produkci. Příklady chyb nalezených v průběhu testování jsou obsaženy v příloze.

Sekce Testování uzavírá uzavírá kapitolu Implementace rozšiřujících funkcionalit. V této kapitole jsem se zabýval detailnějšími, technickými aspekty toho, jak byly rozšiřující funkcionality implementovány a také tím, jaký je jejich vztah k ostatním částem stávajícího systému. Nejprve jsme se věnovali třívrstvé architektuře, kterou jsem do stávajícího systému vnesl pro větší přehlednost a udržitelnost kódu. Následně jsem rozebral jednotlivé vrstvy této architektury a stručně popsali technologie a přístupy, které jsem zde aplikoval. Následovala sekce Bezpečnostní aspekty, jejímž obsahem byl způsob zabezpečení nových funkčností. Nakonec jsem nastínil proces toho, jak byly nové funkčnosti testovány před nasazením.

Kapitola 7

Zhodnocení výsledků

V této poslední kapitole práce shrnu výsledky celého projektu. Nejprve nastíním některé možnosti dalšího směřování stávajícího systému, které se otevírají po implementaci rozšiřujících funkcionalit, jenž byly obsahem této práce. Následně zmíním zpětnou vazbu, kterou jsem dostal od vedení Lingua Nostra, poté vyhodnotím projektovou práci - porovnám předpoklady o tom, kolik času měla zabrat jaká část se skutečnými výsledky - a nakonec uvedu, některé důletité poznatky, které jsem v průběhu této práce získal.

7.1 Další směřování

7.1.1 Refactoring kódu

Tato subsekce se bude zabývat tím, jak se může dále vyvíjet kód již implementovaných funkcionalit. Je to velice důležitá oblast, protože rozšiřování kódu nevhodným způsobem může snadno zhoršit jeho čitelnost a robustnost do té míry, že další rozšíření už nebude reálně možné a nezbude, než systém zahodit a začít od začátku.

První důležitou činností, která nepochybně přispěje k dalšímu zdárnému růstu, je refactoring kódu. I když mnou implementované funkcionality plní svůj účel, jejich kód rozhodně není optimální z hlediska čitelnosti a úspornosti, vyskytují se také duplicity, především jde o validaci formulářů zákazníka a nájemníka. Dále se domnívám, že by refactoringu měl být podroben adresář includes a především soubory ...esentials.php, protože se jedná o antipatterny, jsou to třídy bez jasného účelu, které plní mnoho vzájemně nesouvisejících funkcí. V tomto adresáři by také bylo dobré jasně definovat fungování tříd označených jako DTO - Data Access Object. Některé z nich skutečně plní účel tohoto návrhového vzoru, jak jej popisuje Martin Fowler (Tím je serializace a předávání objektů mezi různými procesy, například ve formátu json nebo xml, viz [6, p. 352]), jiné však slouží zkrátka jako objekty doménového modelu, zejména pro elegantní způsob, jímž jsou instancovány z výsledků SQL dotazu, s čímž úzce souvisí další bod.

Pro lepší čitelnost a zvládání složitosti systému by bylo vhodné jasně oddělit vrstvy aplikace. Vhodným přístupem by mohlo být například transformování současných tříd ...Service (v jejichž případě se, jak bylo již výše uvedeno, o

services nejedná) do controllerů ve smyslu vzoru MVC, přičemž by v nich byla ponechána pouze logika zpracování požadavku a vrácení views, která by se tímto také vydělila do samostatné oblasti. Dále je vhodné vytvořit skutečné services (či rovnou modely), kam by se přesunula aplikační logika v současné době přítomná ve třídách ...Service. V návaznosti na to by měl být konsolidován přístup k databázi, například s využitím mnou již vytvořené a používané DAO vrstvy. To by následně mohlo umožnit zavedení PHP PDO nebo objektově relačního mapování (PHP PDO a ORM jsou zcela různé koncepty, protože PDO poskytuje abstrakci přístupu k databázím a ORMy poskytují mapování databázových tabulek na modelové entity). Tím by v systému vznikly pevně oddělené vrstvy a jasná architektura, což je nepopiratelně pozitivní změna. Není však pochyb o tom, že takováto restrukturalizace by byla pracná a časově náročná, protože by bylo nutné vyčistit i takové části kódu, jako je metoda renderApplicationList (ve třídě ApplicationAdminService.php), která má tři sta padesát dva řádků (!) a mísí se v ní aplikační logika, prezentační logika a přístup k databázi. Domnívám se, že takovéto metody poněkud snižují čitelnost kódu.

V souvislosti s remodellingem controlleru by také bylo vhodné vybudovat RESTful API. Zjednodušeně řečeno, REST API je architekturální styl pro návrh rozhraní aplikace, založené na přístupu ke zdrojům pomocí HTTP. K datům se přistupuje pomocí metod GET, POST, PUT a DELETE, které odkazují ke čtení, vytvoření, změně a smazání dat (v tomto pořadí). Při tvorbě REST API je třeba dbát na konzistenci endpointů, na správnou syntax, která odpovídá definici RESTful, a hlídat si verzování rozhraní, aby nedošlo k invalidaci URL. Výhodou rest api by mimo jiné bylo, že by nám umožnilo zbavit se třídy Ajax.php, která v současné době plní roli pseudo-controlleru, který zpracovává všechny příchozí asynchronní požadavky. (viz [5])

7.1.2 Business aspekty

Posuňme se nyní od změn v kódu k možnostem dalšího směřování z business hlediska. Jednou cestou byla vícekrát zmiňována na konzultacích se zástupci LN, je zapojení gamifikace. Gamifikace je strategie pro zlepšování systémů, služeb, aktivit nebo organizací, kde se pokoušíme zlepšit výsledky učení pomocí zapojení prvků game designu - nebo lakoničtěji, jak uvádí teoretik gamifikace Zachary Fitz Walter: Gamifikace je přidávání herních prvků do neherních činností (viz [10]) Tato strategie je hojně uplatňována v komerční sféře především pro zaškolování a rekvalifikaci zaměstnanců. Mezi příklady takových organizací jsou i významná jména jako Cisco, Deloitte, Samsung, Microsoft nebo Google. (viz [8])

Možná forma zapojení gamifikace do fungování Lingua Nostra by bylo vyvinutí vlastní aplikace, která by fungovala podobně jako aplikace Wordwall, byla by však přizpůsobena na míru Lingua Nostra jak co do funkcionalit tak do vzhledu. Wordwall je webová aplikace, která umožňuje uživatelům vytvářet pomocí předpřipravených šablon studijní aktivity, které si uživatelé buďto tisknou ve formě dokumentů (printables), nebo jsou k dispozici online

jako takzvané Interactives. Takovýto projekt by mohl poskytnout Lingua Nostra oproti ostatním jazykovým školám významnou výhodu.

Jiným aspektem, který by Lingua Nostra mohla v budoucnosti dále rozvíjet, je online výuka. Ze začátku se u Lingua Nostra jednalo o vynucený krok, který byl proti srsti jak lektorům, tak samotným studentům. Důvodů je mnoho: jednak účast na výuce komplikují meze možností technologického zázemí a schopností všech zúčastněných, jednak neosobní kontakt přes webkamery v lepším případě nebo pouze v hlasové podobě v případě horším výrazně snižuje schopnosti se soustředit a obecně zhoršuje zážitek z výuky, nehledě na to, že některé výukové aktivity nejsou v online prostředí zkrátka možné. Na druhou stranu ale online výuka přinesla i pozitiva: šetří čas studentů i lektorů, protože se nemusí fyzicky přesouvat do různých lokalit, které v konkrétním případě Lingua Nostra jsou od sebe někdy značně vzdálené, a také může šetřit peníze, neboť online výuka nevyžaduje pronajatou učebnu. Další důležitou výhodou prezenční výuky je to, že se mohou otevřít i kurzy, které by se pro nedostatek zájemců jinak neotevřely, protože volná místa budou zaplněna mimopražskými klienty, pro které by jinak fyzická vzdálenost byla nepřekonatelnou bariérou.

Mimoto by bylo užitečné do budoucna vylepšit některé čistě technické aspekty fungování webu, jako například zlepšení některých bezpečnostních prvků, automatizovat přechod na nové kalendářní období. S touto agendou se v současné době pojí několik ne zcela triviálních aktivit a automatizace bude šetřit čas. Podobně by bylo vhodné umožnit přidávat z administrativního portálu nové lektory a nové místnosti - toto je zatím řešeno přímo přidáváním řádků do databáze. S využitím nově naimplementované DAO vrstvy by ale přidání těchto funkcionalit bylo relativně jednoduché.

7.2 Zpětná vazba Lingua Nostra k výsledkům projektu

Spolupráce s Lingua Nostra probíhala po lidské stránce velmi dobře, což práci na projektu výrazně usnadňovalo. Zřídka se vyskytly situace, kdy jsme nezvládli organizační stránku spolupráce, ale tyto neměly podstatné následky. Tato práce má důležitý praktický a užitný rozměr, a proto se domnívám, že je na místě zhodnotit i jaký byl její výsledek v tomto ohledu. Kompletní představu budeme mít samozřejmě až po nějaké době, kdy bude rozšíření v provozu a bude je možné zpětně zhodnotit. V tuto chvíli je řešení nasazeno a je spuštěna evidence individuálních studentů. Za sebe mohu subjektivně říci, že to, čeho jsem měl v plánu v této oblasti dosáhnout, je splněno. Po úplnost uvádím také zpětnou vazbu ze strany vedení LN:

Po skoro deseti letech fungování naší školy se nám počet studentů v individuálních a firemních kurzech postupně rozrůstal, přestože naší doménou jsou skupinové jazykové kurzy. Najednou jsme se ocitli v situaci, když nám nestačil stávající systém zadávání odučených lekcí, který spočíval v manuální práci a excelové tabulce. Do tabulky se ručně zadávaly odučené lekce na základě pdf

výkazů lektorů. Podle tabulky se potom ručně fakturovalo. Tento systém při větším počtu studentů a lekcí začínal být silně nevyhovující.

Nyní máme software, do kterého každý lektor zadá na konci měsíce své odučené lekce s individuálními či firemními studenty, naše kancelář má tak okamžitě přehled. V systému jsou odděleny dvě skupiny, studenti, kteří si kupují balíčky, těm se odečítá kredit a, když vidíme, že jim balíček končí, pošleme jim jednoduše faktura za další. Druhá skupina dostává každý měsíc fakturu za odučené lekce v tom daném kalendářním měsíci. V systému stačí párkrát kliknout a faktura odejde, částka je předem jasná a není třeba nic počítat.

Pan Kylberger se s námi několikrát sešel, aby zmapoval dosavadní situaci a pochopil, co bude pro nás to nejlepší. Software pak vypracovat dle našich požadavků a potřeb, sám nás v průběhu upozornil na některé funkce, které bychom mohli potřebovat. Systém jsme již uvedli v provoz a jsme nad míru spokojeni. Chybovost, která dříve byla vysoká, nyní je nulová a ušetříme velké množství času.

7.3 Vyhodnocení projektové práce

Prací na tomto projektu jsem strávil celkem 376 hodin. Z toho naprostou většinu - 280 hodin - zabrala samotná implementace funkcionalit. Dalších 112 hodin bylo věnováno tvorbě textu. Podle původních očekávání měla být implementace funkcionalit správy individuálních studentů hotová relativně rychle a největší část měla zabrat tvorba nájemního portálu. Ve skutečnosti byl poměr přesně opačný. Domnívám se, že je tomu tak proto, že právě v počátečních stadiích projektu jsem se seznamoval s novými technologiemi a orientoval se ve struktuře stávajícího kódu. To způsobilo, že jsem často narážel na technické problémy, a tudíž mi implementace funkcionalit, které byly logicky poměrně jednoduché, zabrala výrazně déle, než jsem očekával. K tomu můžeme navíc připočíst i to, že mnoho částí kódu a jeho struktury jsem při tvorbě nájemního portálu přepoužil, což jeho vývoj značně zrychlilo. Konzultace nebyly ani zdaleka tak časté, jako u semestrálního projektu a zabraly celkem 16 hodin. Jednalo se většinou o řešení konkrétních otázek, často technického charakteru.

Analýza a návrh v rámci semestrálního projektu zabraly celkem 47 hodin. Při zpětném pohledu se to ale jeví jako málo. To, co jsem podcenil nejvíce, bylo seznámení se s technologiemi, které budu používat. Tato fáze v klasickém diagramu vodopádového modelu chybí, a proto je snadné nabýt dojmu, že není příliš podstatná. Tento diagram celkem pochopitelně předpokládá, že ti, kdo budou na projektu pracovat, mají dostatečné zkušenosti a potřebné technologie víceméně znají, v mém případě tomu tak však nebylo. Je otázkou, zda by lepší (a tudíž časově nákladnější) seznámení se s technologiemi na začátku práce celkový čas snížilo, či zda by jej jen přesunulo z jedné položky do jiné. Jsem toho názoru, že i pokud by tomu tak bylo, tak ve výsledku by samotná implementace byla alespoň mentálně méně náročná. Alespoň pro mě osobně je zajímavější řešit problémy související s logikou aplikace, než ty,

které souvisí např. s neznalostí některého jazykového konstruktu.

Činnost	Hodiny	%
Analýza a návrh	47	10,4
Implementace	280	61,5
Konzultace	16	3.5
Text	112	24,6
Celkem	455	100

Tabulka 7.1: Časová náročnost jednotlivých činností

7.4 Lessons learned

Již v tuto chvíli je zjevné, že pro mě tato práce byla velkým posunem vpřed v mnoha ohledech. Prvním a nejméně překvapivým (avšak ani zdaleka ne nejméně důležitým) jsou nově nabyté znalosti technologií. Mnohem lépe se orientuji v PHP, s nímž byla doteď mou jedinou zkušeností semestrální práce na předmět ZWA, naučil jsem se používat stále rozšířený Ajax a JQuery a interagovat s databází pomocí mysqli. Získal jsem alespoň základní znalosti frameworků PHPUnit a Mockery, procvičil jsem si některá úskalí verzovacího systému Git v praxi. Seznámil jsem se technologií Grunt a naučil jsem se používat v současnosti velmi populární Composer (správa závislostí v PHP).

V obecnější rovině jsem si vyzkoušel, jak důležité a přitom komplikované části vývoje softwaru jsou analýza a návrh. Pochopil jsem, že věnovat velké množství času pečlivému návrhu se vyplatí, a to navzdory tomu, že času je vždy příliš málo a čas, kdy nepřibývá kód a nové funkcionality, se zdá být promarněný. S tím souvisí i to, že dobrý návrh není možný, pokud se dobře neseznámíme s technologiemi, které budeme využívat. Pokud budeme schopni dobře využít jejich vlastnosti, dostaneme se k mnohem efektivnějším řešením. Samozřejmě i zde si musíme uvědomit, že má smysl se učit jak obecně funguje PHP, i když máme jen pár týdnů na implementaci většího počtu netriviálních funkcionalit.

Výrazný přínos se pro mě však nacházel v rovině interakce se zákazníkem. Analýzu jsem doteď neměl příležitost si vyzkoušet v praxi, a i když jsme si konzultace se zákazníkem na některých předmětech simulovali, reálná zkušenost byla zcela jiná. Předně, zákazník v mém případě nebyl jedna jediná osoba, z čehož plyne, že představy o výsledku jednotlivých stakeholderů ze strany zákazníka se často do nějaké míry lišily. Konzultoval jsem samozřejmě s více členy LN, nicméně jejich počet byl prostě nedostačující. Další důležitý postřeh je, že je přínosné konzultovat s více uživateli zároveň, protože velmi rychle vyplynou na povrch mnohé důležité nejasnosti a problémy. Až budu příště pracovat na podobném projektu, mám také v úmyslu použít prototyp. Z rozhovorů s uživateli získáme mnoho informací a je přínosné i sledovat je využívat stávající systém, ale pokud budeme mít možnost je pozorovat jak

používají simulaci toho nového, bude snadno patrné, v čem se naše představy o jejich potřebách liší od skutečnosti. Jako součást semestrálního projektu jsem sice vytvořil wireframy, avšak pouze statické a ty se ukázaly jako pro tento účel nedostačující.

Nakonec bych rád zmínil přínos, který byl pro mě osobně nejdůležitější, a sice zkušenost ze zápasu s komplexitou již fungujícího systému. Kód stávajících funkcionalit nebyl vždy přehledný a předvídatelný, zorientovat se v něm trvalo dlouho, a přesto jsem se i v pozdějších fázích vývoje dostával do situací, kdy můj kód nebo představy o tom, jak by měl fungovat, byly v konfliktu s dosavadním stavem. Porozumění bránila jak nedostatečná znalost některých technologií, tak komplexita kódu samotného, tak nedostatek komunikace s IT pracovníkem. Domnívám se, že poslední problém by bylo možné vyřešit větší průbojností, vyvinutí nějaké míry tlaku na objasnění fungování složitějších částí systému, protože by se tím výrazně ušetřil čas, který jsem strávil jejich dešifrováním.

Kapitola 8

Závěr

V této práci jsem v návaznosti na semestrální projekt spolupracoval s jazykovou školou Lingua Nostra. Nejprve jsem provedl analýzu jejího fungování. Popsal jsem stručně její historický vývoj od malé instituce s několika lektory až po dominantního hráče na poli zájmové výuky italského jazyka; její organizaci, přičemž jsem se soustředil na klíčové procesy, které zde probíhají: nejen výuka, ale také evidence plateb, suplování a další.

Následně jsem se zaměřil na ty oblasti jejího fungování, které jsem identifikoval jako problematické a které by bylo možno podpořit pomocí informačního systému. Výstupem zde byla především evidence individuálních studentů, což byl problém, který z Lingua Nostra odčerpával neúměrné množství zdrojů, protože byl řešen již nedostačujícími metodami; využití místností, které v průběhu pandemie výrazně kleslo a mohlo by při správné IT podpoře těžit například z pronájmů externím subjektům; a některé další oblasti, které jsme ale po diskuzi zavrhli.

Poté jsem analyzoval stávající podpůrné systémy a jejich stav. Byla to náročná činnost, neboť systém byl již rozsáhlý a jeho kód byl v některých oblastech špatně čitelný. Komplexnost systému byla taková, že jsme nutně museli zvážit variantu celý systém zahodit a naimplementovat úplně nový, udržitelnější. Výstupem diskuze však bylo rozhodnutí stávající systém rozšířit. Jako prioritní funkčnosti jsme určili evidenci individuálních studentů, nájemní portál a kalendář pro využití místností.

V další části jsem tyto funkčnosti implementoval. Implementace byla náročná, protože bylo potřeba nové funkcionality na jedné straně udržet v souladu se stávajícím kódem a na straně druhé je psát čitelným a přehledným způsobem. Zde bych chtěl ocenit spolupráci IT oddělení, které poskytlo perfektně připravené html šablony a velice cenný vhled do fungování stávajícího kódu.

Po implementaci jsem se zástupci Lingua Nostra provedl uživatelské testování. To bylo úspěšné, ukázalo se, že nové funkcionality splňují očekávání, i když byly odhaleny některé chyby. Žádná z nich však nebyla tak vážná, aby významně negativně ovlivnila fungování systému a rozšíření bylo vzápětí nasazeno. Současný stav se tedy oproti stávajícímu výrazně změnil. Značné množství času, které bylo dříve nutné vynaložit na zdlouhavou a do určité míry chybovou práci s rozsáhlými a komplikovanými excelovými tabulkami

individuálních studentů je nyní možné použít na efektivnější komunikaci s klienty a další agendy. Tato úspora je hlavním přínosem projektu. Využití místností je nyní díky kalendáři přehledné a je možné začít plánovat pronájmy externím subjektům. Stávající systém se může rozvíjet i dál, přičemž jako velmi vhodný se nyní jeví spíše refactoring stávajícího kódu než přidávání nových funkčností, protože pak bude lepší podporou pro případné další rozšíření business ativit.

Cílem této práce (jak je zmíněno v úvodu), bylo pro některou jazykovou školu vyvinout nový nebo rozšířit stávající informační systém tak, aby se výrazně zjednodušila nějaká neúměrně náročná agenda. Jak plyne z výsledků testování a především ze zpětné vazby Lingua Nostra, tento cíl byl splněn.

Literatura

[1] Altexsoft. Acceptance criteria: Purposes, formats, and best practices. "https://www.altexsoft.com/blog/business/acceptance-criteria-purposes-formats-and-best-practices/". (naposledy dostupno: 02.05.2021).

[2] P. v. G. B. Hambling. *User Acceptance Testing: A Step By Step Guide*. Elsevier, 1997.

[3] P. I. Community. Mysql database overview - strengths and weakness of mysql. "https://oracle-patches.com/en/databases/mysql/3284-overview-of-mysql-database?start=2",note="(naposledydostupno:02.05.2021)".

[4] C. S. D. Reiersøl, M. Baker. *PHP in Action*. Manning Publications, 2007.

[5] A. Gillis. Rest api (restful api). "https://searchapparchitecture.techtarget.com/definition/RESTful-API". (naposledy dostupno: 02.05.2021).

[6] M. F. E. H. R. M. R. S. M. Fowler, D. Rice. *Patterns of Enterprise Application Architecture*. John Wiley Sons, Inc., 2002.

[7] K. K. P. Mulvey, K. McGoey. *Business Analysis for Dummies*. John Wiley Sons, Inc., 2013.

[8] B. Patten. Effective training strategies: 7 companies using gamification correctly. "https://blog.insynctraining.com/effective-training-strategies-7-companies-using-gamification-correctly", note="(naposledydostupno:02.05.2021)".

[9] R. Patton. *Software Testing*. Sams Publishing, 2001.

[10] Z. F. Walter. What is gamification? "https://www.gamify.com/what-is-gamification?fbclid=IwAR1NTOTHdVn_vy4u8SCRu2iw1BgUz6UvHxrLReHl_siUCzQUZOymmOytplU". (naposledy dostupno: 02.05.2021).

[11] N. Žunić. Layered architecture. "https://medium.com/java-vault/layered-architecture-b2f4ebe8d587". (naposledy dostupno: 02.05.2021).

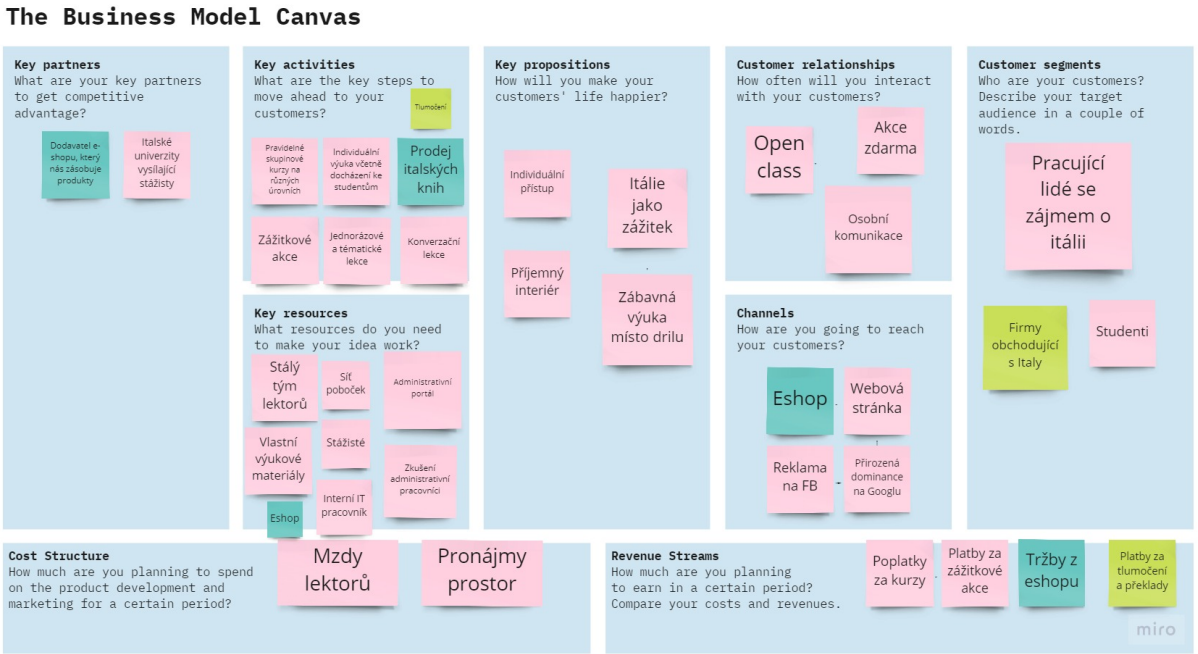
Příloha A

Seznam zkratek

Zkratka	Význam
RPSN	roční procentuální sazba nákladů
LN	Lingua Nostra
CRUD	Create, Read, Update, Delete
DAO	Data Access Object
JWT	JSON web token

Příloha B

Ostatní přílohy



Obrázek B.1: Business Model Canvas

Akceptační kritéria

I1 Pokud individuální student s tímto e-mailem ještě neexistuje, pak když admin zadá validní vstupní hodnoty, bude tento viditelný v seznamu individuálů a bude možno zobrazit jeho profil, kde admin uvidí zadané hodnoty; lektor jej ve výkazu uvidí v dropdownu individuálních studentů.

I2 Pokud je individuální student již vytvořen, pak když mu lektor nebo admin připíše individuální lekci, zobrazí se tato na seznamu individuálních lekcí na profilu studenta a studentův kredit se sníží.

I3 Pokud je individuální student již vytvořen, pak když mu admin vystaví fakturu v zadané výši, bude tato faktura vidět ve Fakturoidu a na studentově seznamu faktur.

I4 Pokud byla individuálnímu studentovi již vystavena faktura, pak pokud ji admin ve fakturoidu označí jako zaplacenou, bude faktura vidět jako zaplacená i na individuálově seznamu faktur a jeho kredit se zvýší.

I5 Pokud byla individuálnímu studentovi již připsána odučená lekce, může ji admin zrušit. Pak se již nebude zobrazovat v individuálově seznamu a jeho kredit se zvýší.

N1 Pokud nájemník s tímto e-mailem ještě neexistuje, pak když admin zadá platné vstupní hodnoty, bude tento viditelný v seznamu individuálů, admin jej uvidí v seznamu, bude moci zobrazit jeho profil, kde uvidí zadané hodnoty.

N2 Pokud je nájemník již vytovřen, pak když zadá nejprve obecné a pak své osobní heslo do přihlašovacího formuláře, je přihlášen a uvidí svůj profil.

N3 Pokud je nájemník již přihlášen, pak když klikne na tlačítko odhlásit, nebude už moci zobrazit ani profil ani kalendář a nájemnický portál od něj bude opět vyžadovat přihlášení.

N4 Pokud je nájemník již přihlášen, pak pokud zobrazí kalendář, uvidí interní sloty bez informací a sloty minulé nebo v následujících 24 hodinách jako nedostupné, své sloty začínající za více než 24 hodin s informacemi jako zrušitelné a své minulé sloty s informacemi, ale bez tlačítka zrušit.

N5 Pokud je nájemník již přihlášen, pak pokud zabookuje nějaké množství slotů, pak je uvidí v kalendáři jako sebou zabookované, zobrazí se mu v seznamu na profilu a zvýší se částka za aktuální měsíc, kterou admin vidí v seznamu.

N6 Pokud je nájemník již vytvořen, pak když mu admin vytvoří fakturu, pak ji nájemník i admin uvidí na nájemnickém profilu v seznamu faktur a pokud je to nejnovější faktura, pak bude zobrazena jako poslední faktura v seznamu nájemníků u jeho jména.

N7 Pokud byla nájemníkovi již vytvořena faktura, pak když ji zaplatí, Fakturoid to zaregistruje a nájemník a admin na nájemnickém profilu uvidí jako zaplacenou, admin ji uvidí jako zaplacenou i v seznamu nájemníků a v seznamu nájemníků se zvýší částka placených faktur za tento měsíc.

N8 Pokud je admin nebo nájemník již přihlášen a v kalendáři označí nějakou skupinu slotů a klikne na "bookovat dopředu," pak pokud vzniknou kolize, prohlížeč je na to upozorní.

A1 Pokud je admin již přihlášen, pak když klikne na kalendář uvidí všechny sloty s informacemi a zabookované sloty bude moci zrušit, sloty odpovídající skupinovým kurzům uvidí pouze s informacemi.

L1 Pokud je lektor již přihlášen, pak pokud vybere množinu slotů a zabookuje je, budou v systému zaneseny jako interní a nájemníci o nich neuvidí žádné informace, admin o nich uvidí informace a bude je moci zrušit, lektor o nich uvidí informace a bude je do 24 hodin před začátkem moci zrušit.

Obrázek B.2: Akceptační kriteria

Testovací strategie

1 C Jedná se o validaci formuláře a třída rizika je nízká, budeme tedy testovat kombinaci vstupů, zvolíme techniku C/DC.

2 B Ukládáme entitu, s níž budeme chtít dále pracovat, budeme tedy testovat životní cyklus objektů, kvůli vyšší prioritě použijeme test datové konzistence s kompletním pokrytím.

3 A Zde potřebujeme pouze, aby lektor viděl individuála v seznamu, dál jej neupravuje, takže toto bude zajištěno testem datové konzistence s kompletní pokrytím z předchozího případu.

4 B Změny kreditu závisí na životním cyklu objektu odučená lekce, budeme tedy testovat životní cyklus dat, a protože je zda maximální priorita, vybereme variantu s úplným pokrytím.

5 B Fakturace individuála se řídí životním cyklem faktury (a to i mimo náš systém tudíž jde o víc než jen třídu InvoiceM), operací je zde relativně málo, takže si můžeme dovolit test s úplným pokrytím.

6 C Opět se jedná o stav faktury a její životní cyklus, toto tedy bude pokryto předchozím scénářem.

7 C Analogicky jako v prvním případě testujeme kombinaci vstupů, opět technika C/DC

8 B Stejně jako v případě 2 zde otestujeme životní cyklus nájemníka. V tomto případě je sice více R a U operací než ve 2, ale vyšší třída rizika toto ospravedlňuje.

9 B Přihlášení a odhlášení jsou z pohledu CRUD matice operace R, proto budou zahrnuty už v testu datové konzistence z případu 2. Toto platí i pro případ 10.

11 B Booking je jedna z komplikovanějších aktivit a lze ji testovat z více různých pohledů. Nejpřínosnější se jeví test datové konzistence. Nemáme zde žádné operace U, proto můžeme použít kompletní pokrytí.

12 C Pro obě varianty bookingu dopředu použijeme kombinace vstupů. Třídy ekvivalence určíme podle toho, kolik bude vybraných slotů, zda jsou ve stejném týdnu, ve stejné učebně, zda nastanou kolize atd. Tento podproces má nízkou prioritu, proto bude stačit technika C/DC.

13 C Zrušení slotu jsme testovali již v rámci bodu 11 jakožto operaci D. Protože má tento subproces nízkou třídu rizika, není nutné přidávat další testy.

14 A Subprocesy 14 a 15 mají nejvyšší třídu rizika. Protože se životním cyklem faktury jsou spojené změny hodnot v seznamu nájemníků, musíme testovat datovou konzistenci s úplným pokrytím.

16 B Zrušení slotu administrátorem probíhá principiálně velmi podobně jako když slot ruší nájemník, použijeme tedy stejnou techniku.

17 C Tento bod a bod 18 jsou opět bookingy, proto můžeme testovat stejně jako v předchozích případech.

Obrázek B.3: Testovací strategie

Bug 1 – Dvojité přidání odučených lekcí	
Souhrn:	Když lektor odešle svůj výkaz, tak se všechny lekce individuálnímu studentovi přidají dvakrát
Detekováno:	24. 4 2021
Závažnost:	Maximální
Detekoval:	Skoupá
Verze:	0.1
Jak reprodukovat:	Vyplnit výkaz jakéhokoli lektora, přičemž přidáme alespoň jednu individuální lekci
Očekávaný výsledek:	Lekce se objeví v seznamu odučených lekcí a sníží se kredit podle součtu jejich cen
Skutečný výsledek:	Lekce se tam objeví dvojmo a snížení kreditu je dvojnásobné, než by mělo být
Poznámky:	Velice závažný bug, musí se opravit okamžitě. Nejspíš je špatně nabindovaný event v JS

Bug 2 – Špatná barva textu kreditu individuála při přehoupnutí přes nulu	
Souhrn:	Když se v admin. portálu sníží kredit individuála pod nulu, zůstane zelený
Detekováno:	24. 4 2021
Závažnost:	Nízká
Detekoval:	Černá
Verze:	0.1
Jak reprodukovat:	V DB nastavíme jakémukoli indivudiálovi kredit na kladnou hodnotu. Pak mu v admin. portálu přidáváme lekce, dokud se nedostane na záporný kredit
Očekávaný výsledek:	Při přechodu kreditu z kladných do záporných hodnot text změní barvu ze zelené na červenou
Skutečný výsledek:	Text zůstane zelený
Poznámky:	Tohle je detail, po refreshi stránky je to už dobře

Bug 3 – Nesouvisející chybová hláška při změně platební metody individuála při prázdné ceně balíčku	
Souhrn:	Když u individuála měníme platební metodu z faktury na balíček, tak se objeví hláška, jako kdyby chybělo IČO
Detekováno:	24. 4 2021
Závažnost:	Střední
Detekoval:	Skoupá
Verze:	0.1
Jak reprodukovat:	Vytvoříme individuála s platební metodou faktura. Pak přejdeme na jeho profil, klikneme na tlačítko editovat, změníme metodu na balíček, cenu necháme prázdnou a klikneme na uložit změny
Očekávaný výsledek:	Objeví se chybová hláška, že je třeba zadat cenu balíčku
Skutečný výsledek:	Objeví se chybová hláška, že IČO je povinné
Poznámky:	Nejspíš jen zapomenutý text hlášky ve validaci, ale je to dost matoucí chyba

Obrázek B.4: Chyby nalezené při UAT