VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

Tomáš Chvostek

Bakalářská práce

Vedoucí práce: doc. Mgr. Jiří Dvorský, Ph.D.

Ostrava, 2023

Č.j.VSB/22/079021
vsboee22026114

Zadání bakalářské práce

Student: **Tomáš Chvostek**

Studijní program: B0613A140014 Informatika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: K2 atmitec s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Mgr. Jiří Dvorský, Ph.D.**

Datum zadání: 01.09.2022

Datum odevzdání: 30.04.2023

Garant studijního programu: doc. Mgr. Miloš Kudělka, Ph.D.

V IS EDISON zadáno: 07.11.2022 12:25:51

Abstrakt

V bakalářské práci popisuji celý průběh odborné praxe, kterou jsem absolvoval ve firmě K2 atmitec s.r.o. Cílem praxe bylo realizovat napojení informačního systému K2 na službu Signi prostřednictvím API a umožnit tak elektronické podepisování dokumentů uložených v tomto informačním systému. Jedná se o jeden z důležitých kroků ke kompletní digitalizaci firem. V první části se věnuji samotné firmě, ve druhé části se věnuji použitým technologiím, ve třetí části se věnuji detailnímu rozboru jednotlivých dílčích úkolů a v poslední čtvrté části se věnuji chybějícím či získaným znalostem a dovednostem a celkovému zhodnocení praxe.

Klíčová slova

odborná praxe, informační systém K2, Delphi, Signi, API

Abstract

In my bachelor thesis I describe the whole course of my professional practice, which I completed in the company K2 atmitec s.r.o. The aim of the internship was to implement the connection of the K2 information system to the Signi service via API and to enable electronic signing of documents stored in this information system. This is one of the important steps towards completing the digitalization of companies. In the first part I discuss the company itself, in the second part I discuss the technologies used, in the third part I discuss the detailed analysis of the individual sub-tasks and in the last fourth part I discuss the missing or acquired knowledge and skills and the overall evaluation of the practice.

Keywords

professional practice, information system K2, Delphi, Signi, API

Poděkování

Rád bych poděkoval mému vedoucímu práce doc. Mgr. Jiřímu Dvorskému, Ph.D. za poskytnuté rady během vypracovávání bakalářské práce. Dále bych chtěl především poděkovat mému oponentovi Ing. Tomášovi Kupčíkovi za veškerou pomoc během praxe, bez něho by tato práce nevznikla. A v neposlední řadě děkuji celému týmu AM2, který mi rovněž během praxe pomáhal.

Obsah

Seznam použitých zkratek a symbolů

AM1	–	Aplikační Moduly 1
AM2	–	Aplikační Moduly 2
QA	–	Quality Assurance
IS	–	Information System
ERP	–	Enterprise Resource Planning
API	–	Application Programming Interface
JSON	–	JavaScript Object Notation
XML	–	Extensible Markup Language
DMS	–	Document Management System
URL	–	Uniform Resource Locator

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Kapitola 1

Úvod

V dnešní době se všichni všechno snažíme digitalizovat. Velkou část tvoří digitalizace dokumentů a především jejich elektronické podepisování. Firma Signi tuto službu poskytuje a jejich řešení nabízí plnohodnotné a legislativou uznatelné elektronické podpisy. V případě digitalizace firem je klíčové, aby takové podepisování umožňovaly. Firmy pro svůj chod v dnešní době běžně využívají ERP systém. Aby firma nemusela spravovat zvlášť dokumenty v ERP systému a zvlášť v Signi, je zapotřebí umožnit propojení právě mezi daným ERP systémem a službou Signi.

Firma K2 atmitec s.r.o. se rozhodla ve své nové verzi informačního systému K2 napojit svůj systém právě na službu Signi. Informační systém K2 umožňuje jednak jednoduchou správu dokumentů, jednak správu dokumentů prostřednictvím DMS a má implementováno jednoduché podepisování dokumentů, tudíž propojení se službou Signi bude alternativou k tomuto stávajícímu podepisování.

O absolvování individuální odborné praxe jsem začal uvažovat hned, když jsem se dozvěděl, že taková možnost existuje. Do firmy K2 atmitec s.r.o. docházím už delší dobu na brigádu, proto jsem se rozhodl na odbornou praxi přihlásit právě tady.

V bakalářské práci se budu postupně věnovat samotné firmě, použitým technologiím, řešením dílčích úkolů a zhodnocením celé praxe.

Kapitola 2

Informace o odborné praxi

V této kapitole se budu věnovat samotné firmě K2 atmitec s.r.o. a mému pracovnímu zařazení během vykonávané praxe.

2.1 K2 atmitec s.r.o.

Firma K2 atmitec s.r.o. byla založena 5. prosince 1991. Jedná se o českou firmu se sídlem v Ostravě zaměstnávající cca 250 lidí. Jejím hlavním produktem je informační systém K2 (dále jen K2), který slouží k řízení podnikových procesů [1].

2.2 Pracovní zařazení

Celé vývojové oddělení má cca 60 zaměstnanců a je rozděleno na čtyři střediska. Středisko Systém má na starost jádro celé K2, středisko QA má na starost testování systému a zbylé dva týmy AM1 a AM2 mají na starost aplikační moduly, které jsou postavené nad jádrem K2. Pod středisko AM1 patří moduly z oblastí nákup, prodej, finance, účetnictví, majetek a marketing. Dále vyvíjí a spravuje K2 API či K2 E-shop. Středisko AM2 se stará o moduly z oblastí logistika, výroba, údržba, reklamace, servis, mzdy, personalistika, projekty a workflow.

Já jsem během praxe pracoval na pozici programátora ve středisku AM2, které je pod vedením Ing. Tomáše Kupčíka. Při řešení úkolů jsem komunikoval především s ostatními členy střediska AM2, ale některé problémy jsem řešil i s kolegy z jiných středisek.

Kapitola 3

Použité technologie

Tato kapitola je věnována technologiím, které jsem během praxe používal. Během praxe jsem převážně použil Delphi, Interní skriptovací jazyk K2 ERP, API a JSON.

3.1 Delphi

Delphi je programovací jazyk a zároveň software, který používá dialekt programovacího jazyka Object Pascal a poskytuje integrované vývojové prostředí pro rychlý vývoj aplikací pro počítače, mobilní zařízení, web atd. V současnosti je Delphi vyvíjeno a spravováno společností Embarcadero Technologies.

Delphi obsahuje editor kódu, vizuální návrhář, integrovaný debugger, komponentu pro kontrolu zdrojového kódu a podporu zásuvných modulů třetích stran. Editor kódu obsahuje funkce Code Insight (doplňování kódu), Error Insight (kontrola chyb v reálném čase) a refaktoring. Vizuální návrhář formulářů má možnost používat buď knihovnu VCL (Visual Component Library) pro vývoj čistě pro systém Windows, nebo framework FMX (FireMonkey) pro vývoj napříč platformami. Klíčovou vlastností je podpora databází, kterou zajišťuje FireDAC (Database Access Components). Delphi obsahuje také základní knihovnu pro běh RTL (Run Time Library). Překladače Delphi generují nativní kód pro Microsoft Windows, macOS, iOS, Android a Linux.

Delphi bylo původně vyvinuto společností Borland jako nástroj pro rychlý vývoj aplikací pro Windows jako nástupce Turbo Pascal. Delphi přidal do stávajícího jazyka plnohodnotné objektově orientované programování a jazyk se rozrostl o podporu generik, anonymních metod, uzávěrů a nativní podporu COM (Component Object Model) [2].

3.2 Interní skriptovací jazyk K2 ERP

Jedná se o vlastní nástavbu Delphi, která využívá vlastní editor i vlastní kompilátor. Skripty je možné vytvářet a spouštět při běhu K2.

3.3 API

API neboli rozhraní pro programování aplikací je soubor definovaných pravidel, která umožňují různým aplikacím vzájemnou komunikaci. Funguje jako zprostředkující vrstva, která zpracovává přenosy dat mezi systémy a umožňuje společnostem otevřít data a funkce svých aplikací externím vývojářům třetích stran [3].

3.3.1 Webové API

Webové API obvykle používají protokol HTTP nebo HTTPS pro zprávy požadavků a poskytují definici struktury zpráv odpovědí. Tyto zprávy s odpovědí mají obvykle podobu souboru XML nebo JSON. Formáty XML i JSON jsou upřednostňovány, protože prezentují data způsobem, se kterým mohou ostatní aplikace snadno manipulovat [4].

3.4 JSON

JSON je standardní textový formát pro reprezentaci strukturovaných dat založených na objektové syntaxi jazyka JavaScript. Běžně se používá k přenosu dat ve webových aplikacích (např. odeslání některých dat ze serveru na klienta, aby se mohla zobrazit na webové stránce, nebo naopak). Přestože se velmi podobá objektové literární syntaxi jazyka JavaScript, lze jej používat nezávisle na jazyce JavaScript a mnoho programovacích prostředí disponuje schopností JSON číst (analyzovat) a vytvářet [5].

Jedná se o univerzální datové struktury a v JSON jsou realizovány s využitím následujících konstrukcí [6]:

- Objekt – neuspořádaná množina párů název/hodnota. Objekt je uvozen levou složenou závorkou a zakončen pravou složenou závorkou. Každý název je následován dvojtečkou a páry jsou pak odděleny čárkou.
- Pole – seřazená kolekce hodnot. Začíná levou hranatou závorkou a končí pravou hranatou závorkou. Hodnoty jsou oddělené čárkou.
- Hodnota – řetězec uzavřený do dvojitých uvozovek, číslo, logická hodnota true nebo false, pole, objekt nebo null.

Kapitola 4

Seznam úkolů zadaných během praxe

Již před nástupem na praxi jsem věděl, co bude náplní mé práce. Celá práce je rozdělená do čtyř hlavních úkolů. V této kapitole se jim budu postupně věnovat a u každého uvedu celý postup řešení a čas, který jsem nad ním strávil.

4.1 Napojení K2 na Signi

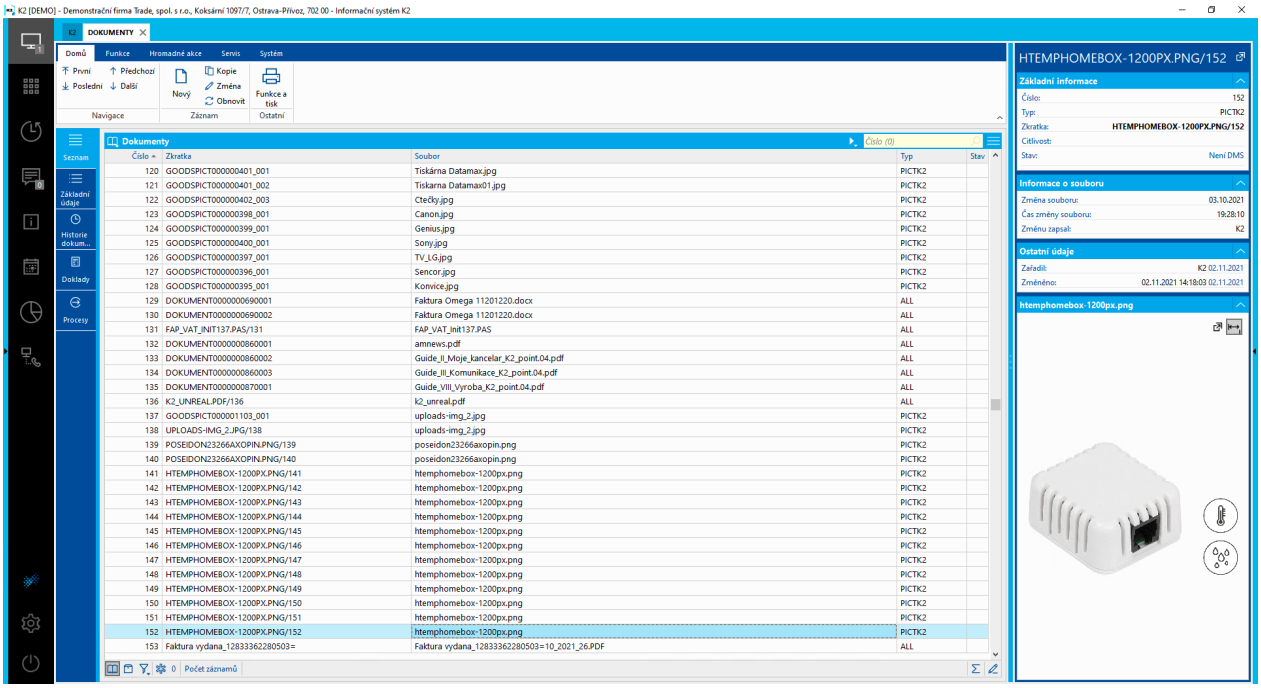
Jako první úkol na praxi jsem měl realizovat napojení K2 na Signi. Nejdříve jsem si udělal analýzu a poté jsem postupně napojení implementoval. Do začátku praxe jsem programoval pouze skripty v jejich interním skriptovacím jazyce, takže zprvu jsem se seznamoval se strukturou samotného systému K2.

4.1.1 K2

Jak jsem již zmiňoval výše, K2 je informační systém pro řízení podnikových procesů (ERP). Hlavním prostředkem pro přístup k datům jsou datové moduly, ve kterých je napsána také veškerá logika informačního systému. Data jsou pro uživatele prezentována prostřednictvím tzv. univerzálních formulářů (jejich vlastní systém formulářů). Ty jsou generované a v případě potřeby je lze snadno upravit přímo v K2 v Návrháři formulářů.

4.1.2 Správa dokumentů v K2

V K2 existuje datový modul Dokumenty, který slouží pro správu veškerých dokumentů v systému. Dokumenty je možné ukládat lokálně, ale K2 poskytuje také svoje řešení DMS a ukládá je do databáze.



Obrázek 4.1: Datový modul dokumentů

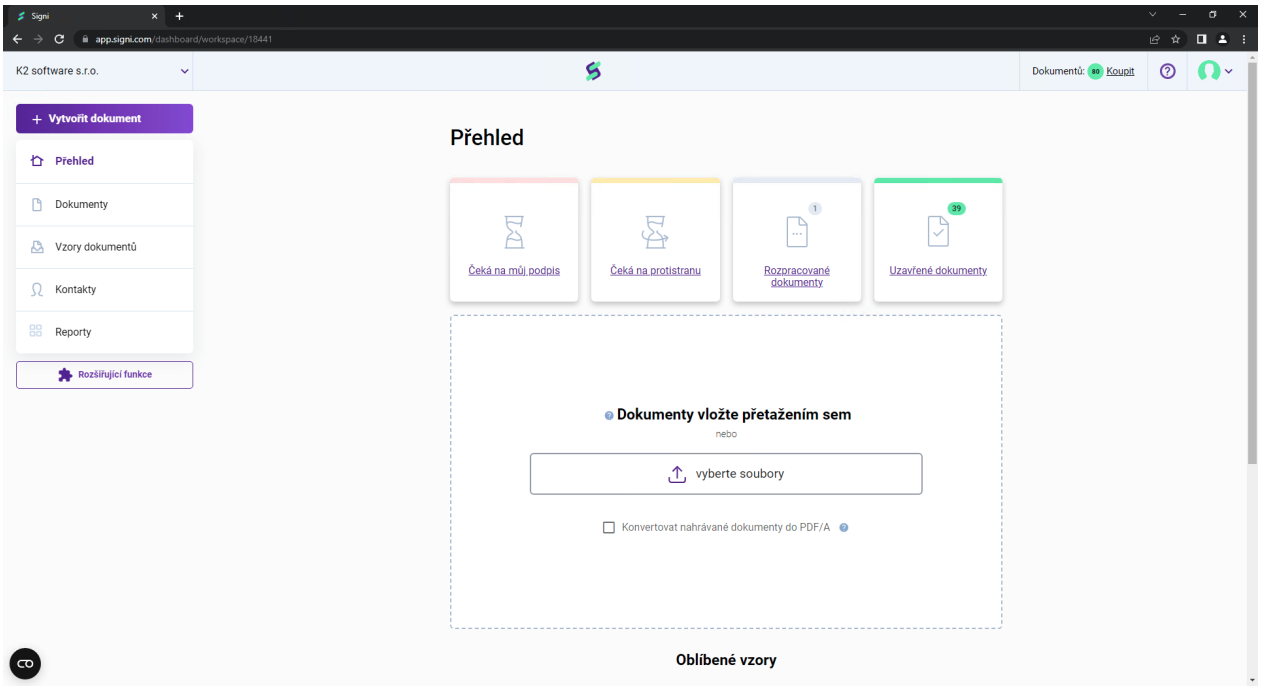
4.1.3 Signi

Signi je česká firma, která poskytuje službu pro elektronické podepisování dokumentů [7]. Vyvíjí vlastní webovou aplikaci, ve které uživatel dokumenty spravuje. Jejich službu mohou využívat fyzické i právnické osoby. Jelikož většina právnických osob využívá některý z dostupných ERP systémů, tak pro napojení jejich služby na tyto systémy mají vyvinuté API.

Pro využívání jejich služeb je nutné mít u nich vytvořený účet. Každý účet má jeden nebo více pracovních prostorů. Tyto prostory slouží především pro rozdělení dokumentů do určitých oblastí a my v nich získáme větší přehled.

Pro odesílání dokumentů slouží kredity, které si na účet dobíjíme. Jeden dokument strhává jeden kredit z účtu. Když chceme dokument odeslat, nahrajeme ho do Signi, vyplníme potřebné údaje, vyplníme jednotlivé podepisující a odešleme. Jeden dokument může zahrnovat až 5 podepisujících. Každý dokument musí mít jednu osobu označenou jako **navrhovatel**. To je osoba, která dokument do Signi nahrává a odesílá, ale může ho i podepisovat. Podepisující jsou **protistrany**.

Po odeslání dokumentu k podpisu Signi odešle podepisujícím email. Jakmile dokument všichni podepíšou, je možné si ho stáhnout a stáhnout si i kontrolní list. To je dokument, ve kterém je zaznamenána veškerá aktivita během podepisování. Uvidíme zde, kdo a kdy dokument podepsal.



Obrázek 4.2: Webová aplikace Signi

4.1.4 Analýza požadavků ze strany Signi

Po společné schůzce vznikl seznam požadavků, které Signi požaduje pro následnou realizaci napojení. Všechny požadavky jsem postupně analyzoval a navrhl jejich řešení.

4.1.4.1 Výběr pracovního prostoru

Pracovní prostor v Signi slouží k rozdělení dokumentů do přehledných skupin. Každý dokument zařazený v Signi patří právě do jednoho pracovního prostoru. Těch můžeme mít vytvořených libovolné množství a při odesílání dokumentu k podpisu si vybíráme, do kterého chceme dokument zařadit. Každý pracovní prostor má svou konfiguraci nastavení.

Návrh řešení Pro konfiguraci jednoho prostoru vytvořím třídu, kterou bude možné zobrazit ve formuláři. Pro správu vytvořených prostorů použiji implementované řešení v K2 pro ukládání vlastních objektů.

4.1.4.2 Výběr jazyka

Signi aktuálně podporuje celkem 13 jazyků, ve kterých je schopno s podepisujícími komunikovat. Jazyk se použije v emailové komunikaci a také pro pokyny při samotném podepisování dokumentu. Výběr jazyka je na úrovni dokumentu, není tedy možné vybrat pro každého podepisujícího jiný jazyk.

Návrh řešení V K2 vytvořím nový číselník.

4.1.4.3 Výběr pořadí podepisování

Signi podporuje 3 možnosti pořadí podepisování dokumentu. Při výběru pořadí *Všichni najednou* se rozešlou emaily všem podepisujícím ve stejný okamžik. Při výběru pořadí *Navrhovatel, pak protistrany* se nejdříve pošle email navrhovateli a čeká se na jeho podpis. Po podepsání se rozešlou emaily protistranám. Poslední volbou pořadí je *Podle pořadí*.

Návrh řešení V K2 vytvořím nový číselník.

4.1.4.4 Výběr typu podpisu

Signi podporuje 5 typů podpisu. Typ se vybírá na úrovni podepisujícího, takže na jednom dokumentu může být kombinace několika typů podpisu. Na výběr jsou typy: *Na vědomí*, *Schválení*, *Biometrický podpis*, *Bankovní identita* a *Podpis certifikátem*.

Návrh řešení V K2 vytvořím nový číselník.

4.1.4.5 Výběr místa podpisu

Signi umožňuje rozmisťovat podpisy na základě 3 možností. První možností je umístit podpis na *Konec dokumentu*. Druhou možností je umístit podpis na zadané *Souřadnice* a třetí možností je umístit podpis na zadaný *Placeholder* neboli speciální pole v dokumentu. Při výběru umístění dle souřadnic nebo speciálního pole bude možné zadat více hodnot.

Návrh řešení V K2 vytvořím nový číselník. Dále vytvořím dvě třídy, které bude možné zobrazit ve formuláři. Jedna třída bude sloužit pro zadávání souřadnic, druhá pro zadávání speciálního pole a z obou bude možné vytvořit kolekce, abychom mohli zadat více hodnot.

4.1.4.6 Konfigurace scénáře podepisování

Jedná se o konkrétní nastavení výše zmíněných parametrů u jednoho dokumentu, který chceme odeslat k podpisu.

Návrh řešení V K2 vytvořím třídu, kterou bude možné zobrazit ve formuláři. Na formuláři budou také tlačítka pro uložení a odeslání. Jakmile uživatel scénář uloží nebo odešle, instance třídy se serializuje do XML formátu a uloží se do existující databázové tabulky XMLItem s propojením k danému dokumentu.

4.1.4.7 Zobrazení stavu dokumentu

Po odeslání dokumentu k podpisu je zapotřebí určit jeho stav, ve kterém se aktuálně nachází. Po odeslání je dokument ve stavu *Odeslaný*. Po podepsání všemi podepisujícími se dokument dostává do stavu *Uzavřený*. Jestliže alespoň jeden podepisující dokument odmítne podepsat, dostává se do stavu *Odmítnutý*. V případě, že se nestihne dokument podepsat, přechází do stavu *Expirovaný*.

Návrh řešení V K2 vytvořím nový číselník a do dokumentů přidám nové nepovinné pole s názvem Stav Signi, které bude odkazovat do nově vytvořeného číselníku. V číselníku bude, kromě výše zmíněných stavů, také stav *Čekající* a ten bude znamenat, že dokument má vytvořený scénář, ale ještě nebyl odeslán k podpisu.

4.1.4.8 Stažení dokumentu do K2

Po podepsání dokumentu všemi podepisujícími bude potřeba dokument stáhnout zpět do K2.

Návrh řešení V K2 chceme mít jak původní dokument, tak podepsaný dokument. Ve většině datových modulů je položkový datový modul dokumentů, do kterého lze přiřadit zařazené dokumenty a propojit je tak s hlavičkovým záznamem. Tento položkový datový modul přidám do datového modulu dokumentů a tím umožním podepsaný dokument propojit s původním dokumentem v K2.

4.1.4.9 Stažení kontrolního listu do K2

Každému dokumentu v Signi se automaticky vytvoří tzv. kontrolní list. Jedná se o samostatný dokument, do kterého se zaznamenává veškerá manipulace s podepisovaným dokumentem. Může se jednat například o otevření dokumentu, podepsání dokumentu nebo odmítnutí dokumentu.

Návrh řešení Řešení bude po vzoru stažení podepsaného dokumentu.

4.1.4.10 Podpis na jednom zařízení

Signi umožňuje podepsat daný dokument ihned po odeslání na stejném zařízení, na kterém odeslání proběhlo. Po odeslání se podepisujícím nerozešlou emaily, ale v prohlížeči se otevřou webové stránky pro podpis.

Návrh řešení Na formuláři konfigurace scénáře podepisování bude tlačítko, které dokument odešle k podpisu a hned poté zobrazí v prohlížeči všechny příslušné webové stránky pro podpis všech podepisujících.

4.1.4.11 Uložený scénář podepisování

Je velice pravděpodobné, že uživatel bude chtít použít stejný scénář podepisování u více dokumentů. Může se jednat např. o smlouvu, kterou podepisují všichni najednou a za navrhovatele podepisuje osoba X na konci dokumentu. Proto bychom měli uživateli umožnit si vytvořit a uložit libovolné množství scénářů a poté už je pouze aplikovat na konkrétní dokumenty. Takovému uloženému scénáři budeme dále říkat jen šablona.

Návrh řešení Pro správu vytvořených šablon použiji implementované řešení v K2 pro ukládání vlastních objektů. Šablona bude sloužit pouze pro předvyplnění scénáře podepisování vybraného dokumentu. Bude tedy možné scénář po aplikování šablony před odesláním libovolně upravit.

4.1.4.12 Hromadné odeslání více dokumentů k podpisu

Jedná se o situaci, kdy si připravím v K2 více než jeden dokument k odeslání a budu je chtít odeslat hromadně všechny najednou.

Návrh řešení V K2 je možné vytvořit hromadnou akci. Jedná se o akci nad označenými záznamy v daném datovém modulu. Pro odeslání tedy vytvořím hromadnou akci, která projde označené dokumenty a odešle je všechny k podpisu.

4.1.5 Analýza Signi API

Po seznámení se se Signi a jejich požadavky jsem se věnoval studiu jejich API. Mají k dispozici rozsáhlou dokumentaci, která mi studování značně ulehčila. Zjišťoval jsem jak autorizovat veškeré požadavky, jaké požadavky můžu sestavovat a jaké odpovědi na ně můžu očekávat.

4.1.5.1 Autorizace požadavků

Autorizace požadavků je na úrovni pracovního prostoru. U každého prostoru je možné si vygenerovat API klíč. Tento klíč se použije pro autorizování požadavků týkajících se manipulace s dokumenty umístěnými v daném prostoru. Umístí se jako textový řetězec do hlavičky požadavku s názvem klíče *x-api-key*.

4.1.5.2 Odeslání dokumentu k podpisu

API požadavek pro odeslání dokumentu k podpisu:

- https://api.signi.com/api/v1/contract/?type=doc

Tělo požadavku musí být typu *multipart/form-data* a musí obsahovat dva klíče. První klíč s názvem *data* je pro soubor ve formátu JSON, ve kterém definujeme scénář podepisování a druhý klíč s názvem *uploaded_file_key* je pro soubor, který je určený k podepisování.

Tělo odpovědi je ve formátu JSON a obsahuje ID dokumentu v Signi, případné přílohy, stav a URL adresy pro všechny podepisující.

4.1.5.3 Vrácení stavu podpisů

Signi podporuje dvě možnosti získání stavu dokumentu. První možnost je založená na jednosměrné integraci s opakovaným dotazováním na stav dokumentu a druhá možnost je založená na dvousměrné integraci s vyvoláním tzv. webhooks.

Jednosměrná integrace

API požadavek pro vrácení stavu podpisů:

- https://api.signi.com/api/v2/contract/{contractId}/signIdentities/list

Požadavek je typu GET a za {contractId} se dosadí ID dokumentu v Signi. Tělo odpovědi je opět ve formátu JSON a obsahuje informace o podepisujících a jejich podpisech.

Tento způsob je jednoduchý na implementaci, ale neumožňuje mít stále aktuální stav dokumentu v K2. Aktualizace stavu dokumentu může probíhat např. tlačítkem na formuláři.

Dvousměrná integrace

Jedná se o způsob, kdy se při odesílání dokumentu předají součástí scénáře tři URL adresy pro každou z těchto hodnot:

- **completed** – podepsáno,
- **rejected** – odmítnuto a
- **expired** – expirováno.

Hned při změně stavu dokumentu se zavolá příslušná URL adresa, která bude obsahovat volání webových služeb K2. Tím se spustí skript, který stav daného dokumentu zaktualizuje. Na URL adresu se pošle POST požadavek a tělo požadavku bude ve formátu JSON a bude obsahovat ID dokumentu v Signi, jeho stav, odkaz na podepsaný dokument a případné přílohy.

Tento způsob má výhodu okamžité aktualizace stavu, dokumenty tak budou v K2 stále aktuální. Pro nasazení tohoto způsobu aktualizace musí mít K2 aktivní server webových služeb a tedy aktivní K2 API.

4.1.5.4 Stažení dokumentu

API požadavek pro stažení dokumentu:

- https://api.signi.com/api/v1/contract/{contractId}/download

Požadavek je typu GET a za {contractId} se dosadí ID dokumentu v Signi. Jako odpověď se nám vrátí podepsaný dokument.

4.1.5.5 Stažení kontrolního listu

API požadavek pro stažení kontrolního listu:

- https://api.signi.com/api/v1/contract/revisionList

Požadavek je typu POST, tělo požadavku je ve formátu JSON a obsahuje ID dokumentu v Signi. Jako odpověď se nám vrátí kontrolní list.

4.1.6 Analýza napojení

Po nastudování možností Signi API jsem vytvořil analýzu samotného napojení. Analýza obsahuje požadované cíle řešení a uživatelský scénář.

4.1.6.1 Požadované cíle řešení

Pro dokument bude možné vytvořit, upravit nebo odstranit scénář podepisování. Po vytvoření scénáře bude možné dokument odeslat k podpisu. Jakmile bude dokument odeslaný, bude možné aktualizovat stav dokumentu a bude možné stáhnout podepsaný dokument a kontrolní list.

Email, jméno a příjmení podepisujícího bude možné vyplnit ručně, nebo vyplnit z vybrané kontaktní osoby (v K2 je datový modul kontaktních osob) a nebo vyplnit z vybraných polí v příslušném datovém modulu.

Budou implementovány obě možnosti aktualizace stavu dokumentu a uživatelé si vyberou, který způsob chtějí použít.

Dokument bude možné odeslat k podpisu jak z datového modulu dokumentů, tak z položkového datového modulu dokumentů. Při odesílání z položkového datového modulu bude možné přebírat podepisujícího z hlavičky dokladu/záznamu. Bude možné vybrat podepisujícího podle zadaného dodavatele/odběratele na dokladu nebo podle kontaktní osoby zadané na dokladu. V případě výběru podle dodavatele/odběratele bude možné ještě zvolit, zda se má email brát z kontaktních informací nebo z kontaktní osoby daného dodavatele/odběratele.

4.1.6.2 Uživatelský scénář

Nad dokumentem kliknu na tlačítko pro vytvoření scénáře podepisování (budu mít možnost vytvořit scénář ze šablony) a otevře se mi formulář. Formulář vyplním a kliknu na tlačítko pro odeslání k podpisu. U dokumentu se mi zobrazí stav *Odeslaný*. Nad dokumentem kliknu na tlačítko pro zobrazení stavu podpisů, které před otevřením formuláře aktualizuje stav dokumentu. Otevře se mi stejný formulář, který ale nepůjde editovat a zobrazí u každého podepisujícího, zda dokument

podepsal. Jestliže se stav během aktualizace změní na *Uzavřený*, stáhne se podepsaný dokument a kontrolní list a připojí se k původnímu dokumentu.

4.1.7 Vytvoření nových typových číselníků

Pro vytvoření nového typového číselníku je zapotřebí v jednotce *T_Texty.pas* vytvořit konstantu a určit číslo nového typu.

```
TXT_Type_SigniLanguage = 675;
```

Zdrojový kód 4.1: Vytvoření konstanty pro nový číselník

Poté jsem pomocí programu K2Localization vygeneroval jednotku *Texts_0675.Localization.pas*, ve které jsou dvě konstanty pro každou možnost. První je pro zkratku a druhá je pro popis. Tyto konstanty slouží poté pro překlady do různých jazyků v případě přepnutí K2 do jiného než českého jazyka.

```
//DefaultText: 'cs'
p0000000001_abbrId = CollectionId + '.p0000000001_abbr';

//DefaultText: 'čeština'
p0000000001_descId = CollectionId + '.p0000000001_desc';
```

Zdrojový kód 4.2: Vytvoření dvou konstant pro jednu hodnotu v číselníku

Tímto způsobem jsem vytvořil číselníky pro:

- Jazyk
- Pořadí podepisování
- Umístění podpisu
- Typ podpisu
- Stav dokumentu
- Volbu podepisujícího
- Email podle (při plnění podepisujícího z hlavičky dokladu/záznamu)
- Email odkud (při plnění podepisujícího z hlavičky dokladu/záznamu)

4.1.8 Změny v dokumentech

Do datového modulu dokumentů přidám dvě nová pole. První pole s názvem SigniContractId bude obsahovat ID dokumentu v Signi. Druhé pole s názvem SigniDocumentStateId bude obsahovat ID stavu, ve kterém se dokument nachází. Nejdříve bude zapotřebí upravit databázovou tabulku, poté do datového modulu přidat vazbu pro pole SigniDocumentStateId a nastavit obě pole pouze pro čtení při zobrazení na formuláři.

4.1.8.1 Úprava databázové tabulky Document

Pomocí programu K2Dictionary jsem do databázové tabulky Document přidal pole SigniContractId a SigniDocumentStateId. Program vytvoří SQL skripty této úpravy pro databázové systémy Microsoft SQL Server a Oracle a zároveň upraví Delphi jednotku *Document.pas* určující strukturu tabulky.

4.1.8.2 Přidání vazby

Veškeré vazby polí se přidávají v proceduře `InternalDefineLinks` a pro vytvoření samotné vazby se používá procedura `AddDynamicLink`. První parametr určuje číslo datového modulu, do kterého se má pole odkazovat, do druhého parametru se předává datové pole, do kterého vazbu přidáváme, třetí parametr určuje povinnost a čtvrtý parametr určuje číslo typu (používá se při odkazování do typového číselníku).

```
AddDynamicLink(
    tbTxt, DataField[DOC_SigniDocumentStateId], False, TXT_Type_SigniDocumentState
);
```

Zdrojový kód 4.3: Přidání vazby do číselníku

4.1.8.3 Nastavení polí pouze pro čtení při zobrazení na formuláři

Vlastnosti datových polí se určují v proceduře `CreateDataFields`. Přidaná pole budou pro uživatele pouze informativní, takže jsem pomocí property `ReadOnlyForUI` znemožnil jejich editaci ve formuláři.

```
DataField[DOC_SigniContractId].ReadOnlyForUI := True;
DataField[DOC_SigniDocumentStateId].ReadOnlyForUI := True;
```

Zdrojový kód 4.4: Nastavení polí pouze pro čtení při zobrazení na formuláři

4.1.9 Vytvoření jednotky

Pro třídy, které budu používat při napojení, jsem vytvořil jednotku s názvem *Signi.Classes.pas*. Jednotku jsem umístil do složky K2DMs a bylo zapotřebí ji zaregistrovat do příslušného hlavičkového souboru.

4.1.10 Vytvoření tříd pro Signi API

Data do Signi API se předávají skrze JSON. Každou třídu jsem vytvořil jako abstraktní a jako potomka třídy `TPersistent`. Tento předek umožňuje objekty přiřazovat. Vytvořil jsem čtyři hlavní třídy:

- `TParameters` (pro požadavek při odesílání dokumentu)
- `TSendSigniDocumentResponse` (pro odpověď po odeslání dokumentu)
- `TSigniDocumentStateResponse` (pro odpověď po získání stavu dokumentu)
- `TSigniCheckListRequest` (pro požadavek při stahování kontrolního listu)

Název pole v JSON formátu se určí přidáním atributu `JSONNameAttribute` nad každou proměnnou ve třídě.

```
[JSONNameAttribute('email')]
FEmail: string;
```

Zdrojový kód 4.5: Přidání atributu JSONNameAttribute

Pro serializaci do JSON souboru jsem využil třídu `TJson` a `TJSONObject`. Statickou metodou `ObjectToJsonObject` třídy `TJson` objekt převedu do `TJSONObject`. Následně metodou `ToString` třídy `TJSONObject` převedu objekt do textového formátu JSON. Tento JSON uložím do souboru pomocí metody `SaveToFile` třídy `TStringList`.

```
procedure TParameters.SaveToFile(aFileName: string);
var
  LJSONObject: TJSOnObject;
  LStringList: TStringList;
  LString: string;
begin
  LJSONObject := TJson.ObjectToJsonObject(
    Self, [joIgnoreEmptyStrings, joIgnoreEmptyArrays]
  );
  LStringList := TStringList.Create;
  try
```

```
    LString := ReplaceStr(
      Trim(LJSONObject.ToString), 'positionsplaceholder', 'positions'
    );
    LStringList.Add(LString);
    LStringList.WriteBOM := False;
    LStringList.SaveToFile(aFileName, TEncoding.UTF8);
  finally
    FreeAndNil(LStringList);
    FreeAndNil(LJSONObject);
  end;
end;
```

Zdrojový kód 4.6: Implementace metody SaveToFile

Pro deserializaci z JSON souboru jsem použil statickou metodu `JsonToObject` třídy `TJson`. Metoda je generická a vrací instanci dané třídy.

```
LResponse := TJson.JsonToObject<TSendSigniDocumentResponse>(LResult);
```

Zdrojový kód 4.7: Deserializace z JSON souboru

4.1.11 Správa pracovních prostorů

Konfigurace jednoho pracovního prostoru bude obsahovat:

- **API klíč**
- **Typ elektronické adresy**, který bude odkazovat do existujícího číselníku typů elektronických adres. K2 má datový modul kontaktních osob, ze kterých bude možné vybírat podepisující. Každá kontaktní osoba může mít několik elektronických adres a u každé z nich je potřeba určit typ. Jedna kontaktní osoba může mít pouze jednu adresu daného typu. Podle tohoto typu se použije konkrétní emailová adresa vybrané kontaktní osoby.
- **Typ dokumentu pro uložení**, který bude odkazovat do existujícího číselníku typů dokumentů. Při zařazování dokumentu do datového modulu dokumentů je potřeba určit jeho typ. Tento typ se bude používat při zařazování podepsaných dokumentů a kontrolních listů.
- **URL adresu serveru webových služeb**, která se bude používat při aktualizaci stavu dokumentů pomocí webhooks.
- **Vlastní firmu**, která bude odkazovat do existujícího číselníku vlastních firem. Při vytváření scénáře podepisování se rovnou předvyplní pracovní prostor podle aktuálně vybrané firmy.

4.1.11.1 Vytvoření třídy

Třídu `TSigniWorkspace` jsem vytvořil jako potomka třídy `TUFSmartPersistent`. Tento předek umožňuje objekt zobrazit ve formuláři.

```
TSigniWorkspace = class(TUFSmartPersistent)
private
  FApiKey: string;
  FCompanyId: integer;
  FDocumentTypeId: integer;
  FElectronicAddressTypeId: integer;
  FSWSUrl: string;
published
  [FieldDescription('API klíč')]
  [RequiredAttribute]
  property ApiKey: string read FApiKey write FApiKey;
  [FieldDescription('Vlastní firma')]
  [K2LinkAttribute(tbCompanyRef, [faNoRequired])]
  property CompanyId: integer read FCompanyId write FCompanyId;
  [FieldDescription('Typ dokumentu pro uložení')]
  [K2LinkAttribute(tbDocumentConfiguration, [])]
  property DocumentTypeId: integer read FDocumentTypeId write FDocumentTypeId;
  [FieldDescription('Typ el. adresy')]
  [K2LinkAttribute(tbEAddressType, [])]
  property ElectronicAddressTypeId: integer
    read FElectronicAddressTypeId write FElectronicAddressTypeId;
  [FieldDescription('Server webových služeb - url')]
  property SWSUrl: string read FSWSUrl write FSWSUrl;
public
  procedure Assign(Source: TPersistent); override;
  constructor Create(AOwner: TPersistent); override;
end;
```

Zdrojový kód 4.8: Třída TSigniWorkspace

Třída implementuje poděděnou proceduru `Assign`, ve které jsem určil, jak se má objekt chovat při přiřazování. V konstruktoru třídy přiřadím property `CompanyId` ID firmy, ve které uživatel pracuje.

```
constructor TSigniWorkspace.Create(AOwner: TPersistent);
begin
```

```
  inherited;
  CompanyId := GetCompanies.Current.CompanyId;
end;
```

Zdrojový kód 4.9: Implementace konstruktoru třídy TSigniWorkspace

Použité atributy Názvy polí na formuláři je potřeba určit pomocí atributu `FieldDescription`. Tento atribut má jeden parametr a to je název. Pokud chceme, aby pole bylo povinné, přidáme atribut `RequiredAttribute`, který nemá žádné parametry. Jestliže chceme přidat odkaz do jiného číselníku, přidáme atribut `K2LinkAttribute`. Ten má jako první parametr číslo datového modulu a druhý parametr pole s nastavením vlastností dané vazby.

4.1.11.2 Ukládání pracovních prostorů

Třídu `TSigniWorkspaceConfigAdaptable` jsem vytvořil jako potomka třídy `TCustomAdaptable`. Tento předek je potomkem třídy `TUFSmartPersistent` a zároveň implementuje rozhraní s názvem `IAdaptable`, které slouží pro objekty, které chceme ukládat do databáze a obsahuje tedy například property `ObjectID`, `CaptionLng`, `DescriptionLng` atd. Do vytvořené třídy jsem přidal property `Configuration`, která je typu `TSigniWorkspace`.

```
[ClassCaption('Signi - workspace')]
TSigniWorkspaceConfigAdaptable = class(TCustomAdaptable)
private
  FConfiguration: TSigniWorkspace;
  function GetConfiguration: TSigniWorkspace;
public
  destructor Destroy; override;
  procedure Assign(Source: TPersistent); override;
published
  property Configuration: TSigniWorkspace
    read GetConfiguration write FConfiguration;
end;
```

Zdrojový kód 4.10: Třída TSigniWorkspaceConfigAdaptable

Pro zobrazení uložených pracovních prostorů použiji statickou metodu `ShowObjectList` třídy `TPresenterViewObjectAdapters`. Metoda má čtyři parametry. Jako druhý parametr se posílá název třídy. Podle tohoto parametru se na formuláři zobrazí pouze instance této třídy. V tomto formuláři můžeme záznamy přidávat, odstraňovat nebo editovat. Formulář v K2 již existuje a nebude potřeba ho upravovat.

```
TPresenterViewObjectAdapters.ShowObjectList(
    aAction.ViewRealizer, TSigniWorkspaceConfigAdaptable.ClassName, '', nil
);
```

Zdrojový kód 4.11: Zobrazení pracovních prostorů

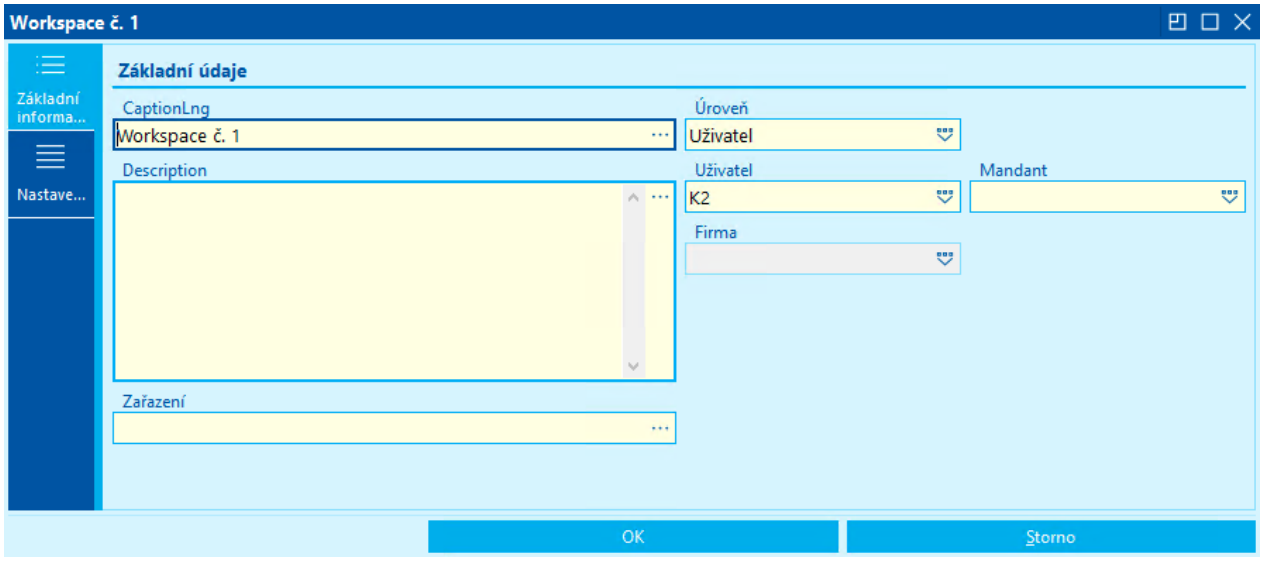
Abychom načetli konkrétní instanci, můžeme použít statickou metodu `LoadNewObject` třídy `TModelUtils`. Generickým parametrem určíme, o jaký typ se jedná. Metoda má poté dva parametry. První parametr je GUID záznamu a druhý (výstupní) parametr je proměnná, do které se uloží získaná instance. Metoda vrací `True`, pokud instanci našla, nebo `false`, pokud instanci nenašla.

```
TModelUtils.LoadNewObject<TSigniWorkspaceConfigAdaptable>(aUI, LConfig);
```

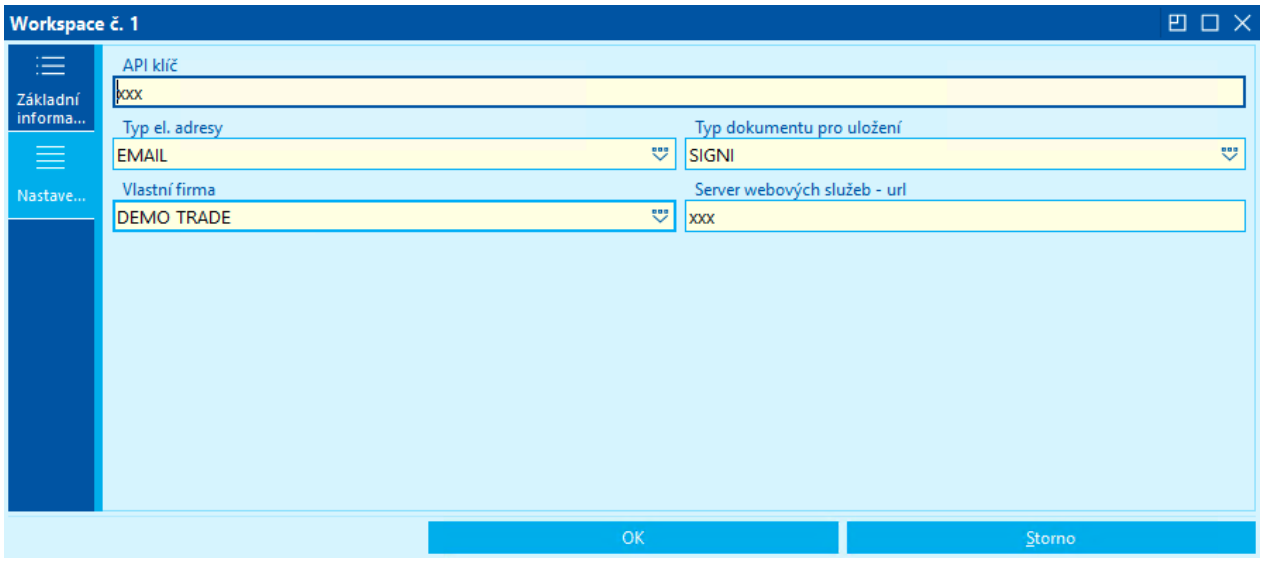
Zdrojový kód 4.12: Získání konkrétního pracovního prostoru

4.1.11.3 Uživatelské rozhraní

Formuláře v K2 jsou generované a lze je jednoduše upravit v K2 v Návrháři formulářů. Pro zobrazení seznamu konfigurací je již v K2 vytvořený formulář, já jsem tedy upravil akorát formulář pro třídu `TSigniWorkspaceConfigAdaptable` tak, že na první záložce Základní informace jsou informace o dané konfiguraci pracovního prostoru a na druhé záložce Nastavení je samotná konfigurace pracovního prostoru.



Obrázek 4.3: Formulář pro pracovní prostor – Základní informace



Obrázek 4.4: Formulář pro pracovní prostor – Nastavení

4.1.12 Umístění podpisu podle souřadnic

Jednou z možností, jak určit místo pro podpis, je předat ve scénáři souřadnice. Pro jednotlivé podepisující se předají souřadnice a může jich být i více než jedny. Pro souřadnice budeme evidovat:

- **Číslo stránky**
- **X**
- **Y**

Vzdálenost X a Y se bere jako vzdálenost levého horního rohu podpisu od levého horního rohu dokumentu v procentech k celkovému rozměru dokumentu.

4.1.12.1 Vytvoření třídy

Třídu `TPlaceholderItem` jsem vytvořil jako potomka třídy `TUFSmartItem`. Tento předek umožňuje jak zobrazit objekt ve formuláři, tak vytvořit kolekci objektů.

```
[ImplicitColumnsAttribute('Page,X,Y')]
TCoordinatesItem = class(TUFSmartItem)
private
  FPage: integer;
  FX: integer;
  FY: integer;
published
  [FieldDescription('Číslo stránky')]
```

```
  property Page: integer read FPage write FPage;
  [FieldDescription('X')]
  property X: integer read FX write FX;
  [FieldDescription('Y')]
  property Y: integer read FY write FY;
public
  procedure Assign(Source: TPersistent); override;
end;
```

Zdrojový kód 4.13: Třída TCoordinatesItem

Použité atributy Použil jsem atribut `ImplicitColumnsAttribute`, který určuje, jaké sloupce se mají zobrazit při zobrazení objektů v tabulce. Tento atribut má jeden parametr a to názvy sloupců v textovém řetězci oddělené čárkou.

4.1.12.2 Vytvoření kolekce

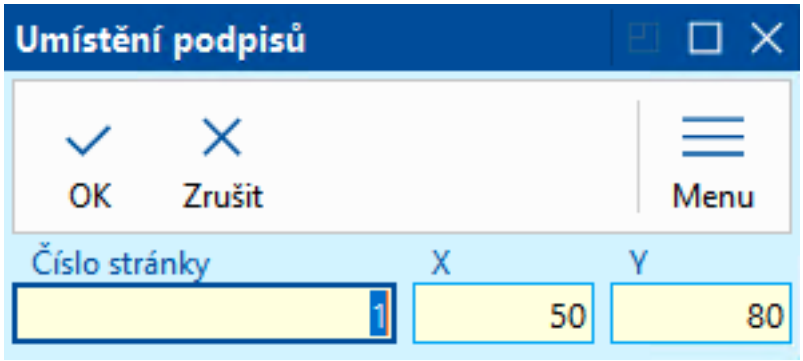
Pro určení více souřadnic u jednoho podepisujícího jsem vytvořil třídu `TCoordinates`, která dědí z generické třídy `TUFSmartCollection`, které jsem předal třídu `TCoordinatesItem`. Tento předek umožňuje kolekci objektů zobrazit ve formuláři.

```
TCoordinates = class(TUFSmartCollection<TCoordinatesItem>)
end;
```

Zdrojový kód 4.14: Třída TCoordinates

4.1.12.3 Uživatelské rozhraní

Pro třídu `TCoordinatesItem` jsem upravil generovaný formulář.



Obrázek 4.5: Formulář pro souřadnice

4.1.13 Umístění podpisu na placeholder

Další možnost, jak určit místo pro podpis, je předat ve scénáři tzv. placeholder. Placeholder je pole umístěné v dokumentu, do kterého se následně vloží příslušný podpis. Ve scénáři se předávají textové řetězce názvů těchto polí. Každý podepisující může mít opět více než jeden placeholder. Pro placeholder budeme evidovat:

- **Placeholder**

4.1.13.1 Vytvoření třídy

Třídu `TPlaceholderItem` jsem vytvořil opět jako potomka třídy `TUFSmartItem`.

```
[ImplicitColumnsAttribute('Placeholder')]
TPlaceholderItem = class(TUFSmartItem)
private
  FPlaceholder: string;
published
  [FieldDescription('Placeholder')]
  property Placeholder: string read FPlaceholder write FPlaceholder;
public
  procedure Assign(Source: TPersistent); override;
end;
```

Zdrojový kód 4.15: Třída TPlaceholderItem

4.1.13.2 Vytvoření kolekce

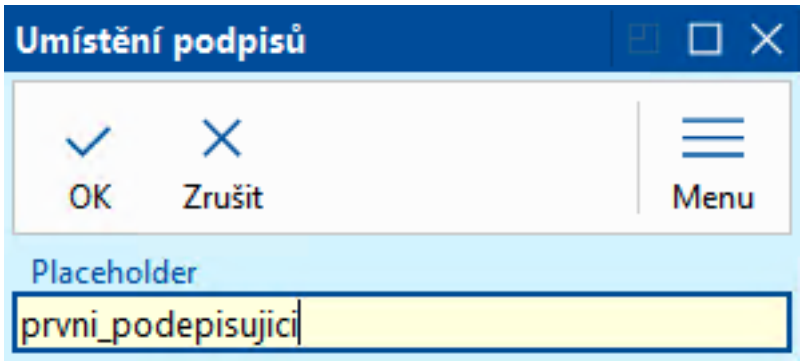
Pro určení více polí u jednoho podepisujícího jsem vytvořil třídu `TPlaceholders` stejným způsobem, jako jsem vytvořil třídu `TCoordinates`.

```
TPlaceholders = class(TUFSmartCollection<TPlaceholderItem>)
end;
```

Zdrojový kód 4.16: Třída TPlaceholders

4.1.13.3 Uživatelské rozhraní

Pro třídu `TPlaceholderItem` jsem upravil generovaný formulář.



Obrázek 4.6: Formulář pro placeholder

4.1.14 Konfigurace podepisujícího

Konfigurace jednoho podepisujícího bude obsahovat:

- **Volbu podepisujícího**, která bude obsahovat čtyři možnosti a bude určovat, odkud se má plnit email, jméno a příjmení. Na výběr budou tyto možnosti:
  - **Bez kontaktní osoby**, hodnoty se budou vyplňovat ručně.
  - **Kontaktní osoba**, hodnoty se budou plnit z vybrané kontaktní osoby.
  - **Podle polí na formuláři**, hodnoty se budou plnit z určených polí na formuláři.
  - **Z hlavičky dokladu** - hodnoty se budou plnit z hlavičky dokladu/záznamu.
- **Kontaktní osobu**
- **Typ elektronické adresy**
- **Pole pro email**
- **Pole pro jméno**
- **Pole pro příjmení**
- **Email podle**
- **Email odkud**
- **Email**
- **Jméno**
- **Příjmení**
- **Navrhovatel**

- **Typ podpisu**
- **Umístění podpisu**
- **Souřadnice**
- **Placeholder**
- **Podepsáno**

4.1.14.1 Vytvoření třídy

Třídu `TSignatoryPerson` jsem vytvořil jako potomka třídy `TUFSmartItem`. Ze třídy `TUFSmartItem` jsem implementoval dvě virtuální funkce.

```
function InternalGetIsRequiredField(const aField: IField): Boolean; override;
function InternalValidateInstance(var aMessage: string): Boolean; override;
```

Zdrojový kód 4.17: Deklarace metod InternalGetIsRequiredField a InternalValidateInstance

Funkce `InternalGetIsRequiredField` má návratový typ `Boolean`. Návratová hodnota určuje, zda je pole povinné. Funkce se volá při jakékoliv změně na formuláři pro všechna pole.

Funkce `InternalValidateInstance` se volá při ukládání instance. Tím umožňuje napsat vlastní validaci na správnost zadaných hodnot. V rámci funkce kontroluji, zda má email správný formát a v případě volby umístění podpisu podle souřadnic kontroluji, zda jsou alespoň jedny souřadnice zadány. Stejná validace probíhá v případě umístění podpisu na placeholder. Jestliže je validace úspěšná, volám funkci předka. Funkce má jeden výstupní parametr `aMessage`, do kterého v případě selhání validace přiřadím informativní zprávu uživateli, která se následně zobrazí v dialogovém okně.

4.1.14.2 Vytvoření kolekce

Pro určení více podepisujících u jednoho dokumentu jsem vytvořil třídu `TSignatory` stejným způsobem, jako jsem vytvořil třídy `TCoordinates` a `TPlaceholders`.

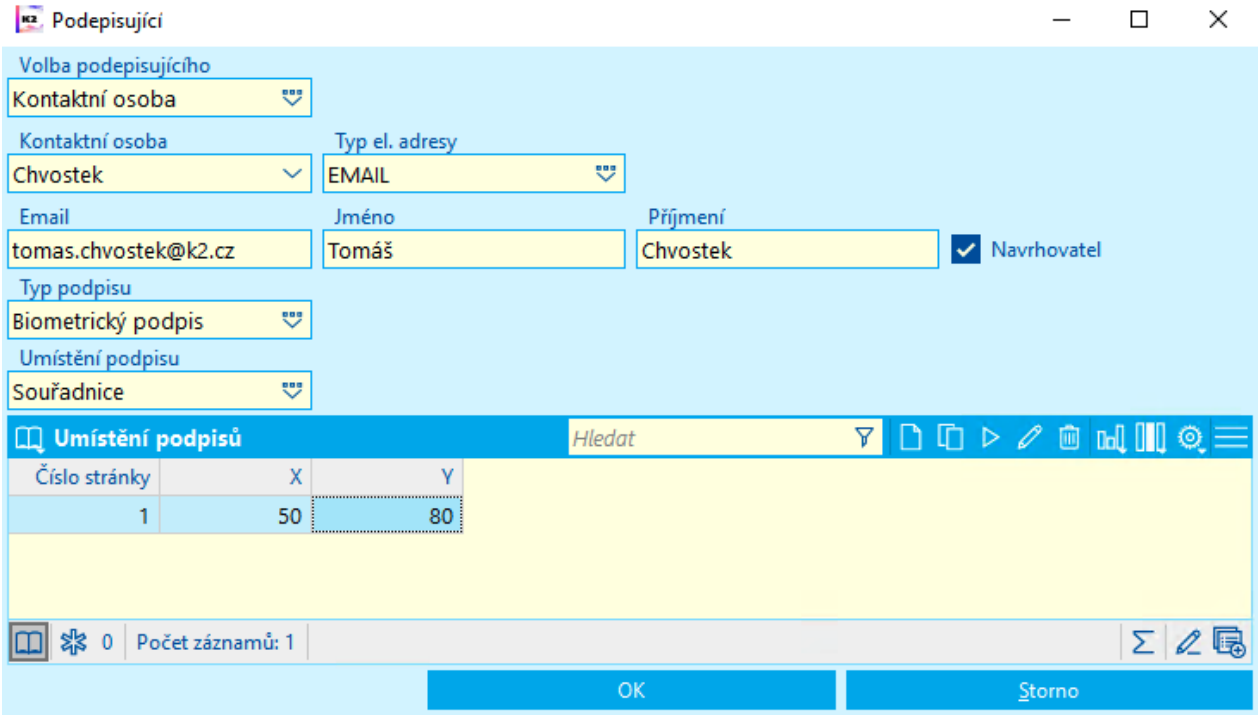
```
TSignatory = class(TUFSmartCollection<TSignatoryPerson>)
protected
  function InternalValidateInstance(var aMessage: string): Boolean; override;
end;
```

Zdrojový kód 4.18: Třída TSignatory

Ze třídy `TUFSmartCollection` jsem implementoval funkci `InternalValidateInstance`, ve které kontroluji, zda je zadán alespoň jeden podepisující a zda je mezi podepisujícími právě jeden navrhovatel.

4.1.14.3 Uživatelské rozhraní

Pro třídu `TSignatoryPerson` jsem upravil generovaný formulář.



Obrázek 4.7: Formulář pro podepisující osobu

4.1.15 Konfigurace scénáře podepisování

Konfigurace scénáře podepisování bude obsahovat:

- **Dokument**, který bude odkazovat do existujícího číselníku dokumentů. Jedná se o dokument určený k podpisu (pro uživatele pouze pro čtení).
- **Pracovní prostor**, který bude odkazovat do existujícího číselníku pracovních prostorů.
- **Jazyk**, který bude odkazovat do existujícího číselníků jazyků.
- **Pořadí podepisování**, které bude odkazovat do existujícího číselníků pro pořadí podepisování.
- **Podepisující**

4.1.15.1 Vytvoření třídy

Třídu `TSigniTemplate` jsem vytvořil jako potomka třídy `TUFSmartPersistent`. Z předka jsem implementoval dvě virtuální funkce.

```
function InternalGetCanEdit: Boolean; override;
function InternalSave: Boolean; override;
```

Zdrojový kód 4.19: Deklarace metod InternalGetCanEdit a InternalSave

Funkce `InternalGetCanEdit` má návratový typ `Boolean`. Návratová hodnota určuje, zda je možné instanci editovat. Před odesláním dokumentu je možné konfiguraci editovat, po odeslání už editace není možná a zobrazuje se pouze stav jednotlivých podpisů.

Návratová hodnota funkce `InternalSave` určuje, zda se podařilo instanci uložit. V této funkci ukládám konfiguraci a měním stav dokumentu na Odeslaný.

4.1.15.2 Ukládání šablon

Ukládání šablon jsem implementoval stejným způsobem jako ukládání pracovních prostorů. Vytvořil jsem třídu `TSigniTemplateConfigAdaptable`, která je potomkem třídy `TCustomAdaptable`.

4.1.15.3 Vytvoření commands

Každá třída v K2 může mít implementované tzv. commands. Na command lze napojit metodu, která se vyvolá stisknutím tlačítka na formuláři nebo stisknutím definované klávesové zkratky. Každý command je potřeba zaregistrovat do jednotky *UniForm.Commands.pas*. Nejdříve se vytvoří konstanta pro název a poté se command zaregistruje pomocí statické metody `Command` třídy `TCmdReg`. Metodou `SetUI` nastavíme ikonu při zobrazení na formuláři a metodou `SetHotKey` nastavíme klávesovou zkratku.

```
cmdSigniSaveDocumentConfiguration = 'SigniSaveDocumentConfigurationCOMMAND';

TCmdReg.Command(cmdSigniSaveDocumentConfiguration)
       .SetUI(trCommitChangeId, glidCommitChange)
       .SetHotKey('F2', [cuisDetail]);
```

Zdrojový kód 4.20: Command – vytvoření a zaregistrování

Napojení metody na command se realizuje přidáním atributu `Command`, který má dva parametry. První parametr je vytvořená konstanta pro command a druhý parametr je název metody, která se má vykonat při každé změně instance a může nastavovat vlastnosti jako např. viditelnost.

```
[Command(
    cmdSigniSaveDocumentConfiguration, 'SigniSaveDocumentConfiguration_Update'
)]
procedure SigniSaveDocumentConfiguration_Execute(aAction: IDataActionExecute);
procedure SigniSaveDocumentConfiguration_Update(aAction: IDataActionUpdate);
```

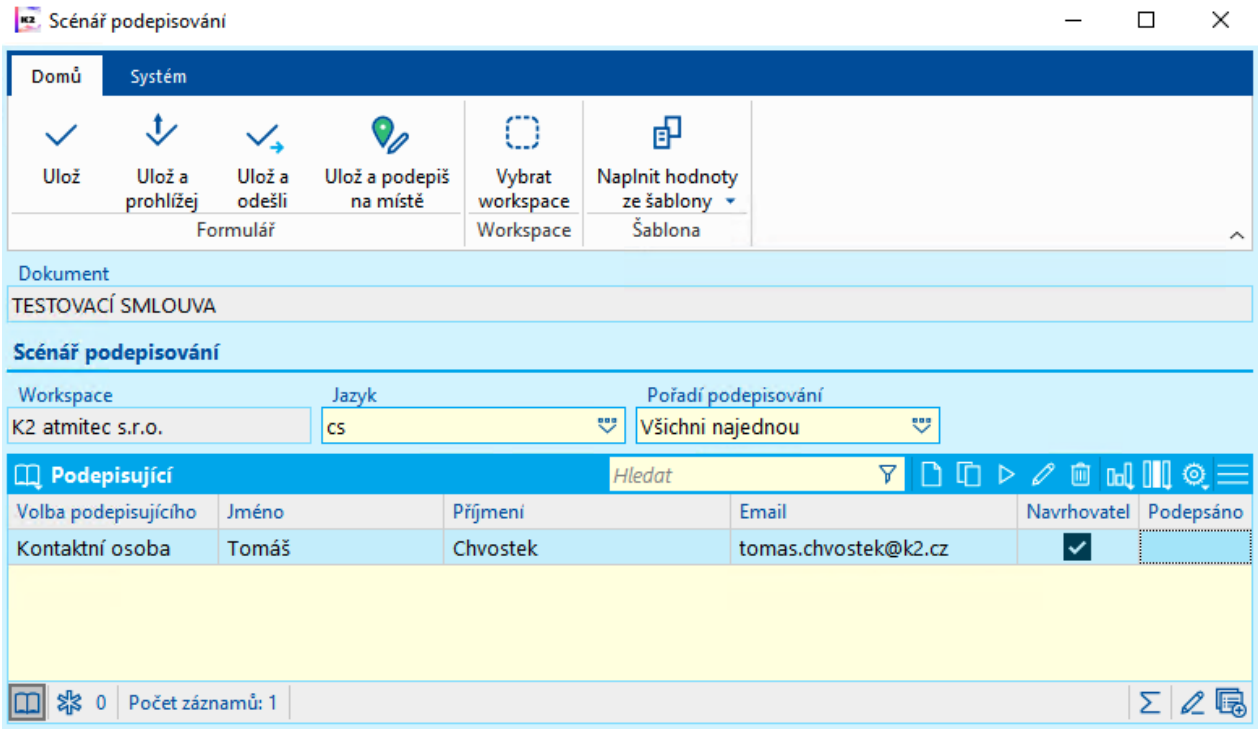
Zdrojový kód 4.21: Napojení metody na command

Tímto způsobem jsem pro třídu `TSigniTemplate` vytvořil následující commands:

- Uložení konfigurace
- Uložení konfigurace a zavření formuláře
- Uložení konfigurace a odeslání dokumentu k podpisu
- Uložení konfigurace a podepsání dokumentu na místě
- Výběr pracovního prostoru
- Naplnění hodnot ze šablony

4.1.15.4 Uživatelské rozhraní

Pro třídu `TSigniTemplate` jsem upravil generovaný formulář.



Obrázek 4.8: Formulář pro konfiguraci scénáře podepisování

4.1.16 Vytvoření commands v presenters

Každý presenter je napojený na datový modul a v něm se vytvářejí commands. Já jsem vytvořil 11 nových commands, které jsem následně napojil na dva presenters. První *Presenter.TD_Dkhl.pas* je napojený na datový modul dokumentů a druhý *Presenter.TD_ExternalDocumentItems.pas* je napojený na položkový datový modul dokumentů. Vytvořené commands:

- Vytvořit scénář bez šablony
- Vytvořit scénář se šablonou
- Upravit scénář
- Odstranit scénář
- Odeslat dokument k podpisu
- Podepsat dokument na místě
- Zobrazit stav dokumentu
- Stáhnout podepsaný dokument
- Stáhnout kontrolní list
- Spravovat pracovní prostory
- Spravovat šablony

4.1.16.1 Command pro hromadné odeslání označených dokumentů

Hromadný command se vytvoří pomocí atributu `CommandBulk`, který má stejné parametry jako `Command`. Tento atribut ale navíc zajišťuje, že command lze spustit pouze nad označenými záznamy a napojená metoda se vykoná pro každý záznam zvlášť. V *Presenter.TD_Dkhl.pas* jsem vytvořil metodu `BulkSendSigniDocument_Execute`, která dokument, který je ve stavu *Čekající*, odešle do Signi. Tuto metodu jsem napojil na hromadný command přidáním výše zmíněného atributu.

4.1.17 Aktualizace stavu dokumentu pomocí webhook

Pro aktualizaci stavu dokumentu bez nutnosti posílat požadavek na Signi jsem vytvořil v datovém modulu dokumentů metodu `SetSigniDocumentStateFromDM` a atributem `ExposeToScript` jsem ji zveřejnil pro skript. Metoda nastaví stav dokumentu dle zadaného parametru.

Dále jsem vytvořil skript *SigniWebhook.pas*, který má parametr `aDocument` pro ID dokumentu a `aStateId` pro stav dokumentu. V těle skriptu se pouze načte konkrétní dokument dle ID a aktualizuje se mu výše zmíněnou metodou stav.

V případě, že uživatel do pracovního prostoru zadá URL adresu serveru webových služeb, při odesílání dokumentu k podpisu se do scénáře vloží tři URL adresy. Každá z nich bude obsahovat volání výše zmíněného skriptu s požadovanými parametry skrze K2 API. Parametr x-auth slouží pro autentizaci požadavku. Konkrétní URL zavolá Signi při přechodu dokumentu do jiného stavu.

```
URL/Formation/raw/Standard/SigniWebhook/pas?aDocumentId=%d&aStateId=%d&x-auth=
```

Zdrojový kód 4.22: Aktualizace stavu dokumentu pomocí webhook

4.1.18 Strávený čas

Dílčí úkol	Počet hodin
Analýza požadavků	15
Analýza Signi API	15
Změny v dokumentech	5
Implementace	190

Tabulka 4.1: Strávený čas nad úkolem Napojení K2 na Signi

4.2 Implementace do DMS

Do této chvíle jsem pracoval pouze s dokumenty, které nebyly uložené v databázi. Odesílal jsem dokumenty, které byly zařazené v K2, ale byly uložené lokálně v počítači uživatele. V případě aktivace DMS a jeho nastavení se vytvoří dvě samostatné databáze. První eviduje veškeré změny či manipulace s dokumentem a druhá eviduje archivované dokumenty. Nepotřebné dokumenty je možné skartovat.

4.2.1 Analýza

Každý dokument zařazený v K2 je v jednom z těchto stavů:

- *Není DMS*
- *Normální*
- *V mé revizi*
- *V cizí revizi*
- *Archivován*
- *Skartován*

Stav *Není DMS* označuje dokument, který není uložen v databázi. Stav *Normální* označuje dokument, který je v databázi a nikdo ho aktuálně nemodifikuje. Stavy *V mé revizi* a *V cizí revizi* označují dokument, na kterém aktuálně někdo pracuje a může ho libovolně modifikovat. Stav *Archivován* označuje dokument, který je archivovaný a nelze ho tedy upravovat. Takový dokument je možné přesunout z archivu zpět a opět umožnit jeho modifikaci. Stav *Skartován* označuje dokument, který už v databázi neexistuje a není možné ho obnovit.

Před odesláním dokumentu do Signi budu kontrolovat, zda je dokument ve stavu *Není DMS* nebo *Normální*. V případě, že je dokument v jiném stavu, nepovolím uživateli dokument odeslat. Jestliže bude dokument skartován, nebude možné aktualizovat jeho stav a ani stahovat podepsaný dokument nebo kontrolní list.

Po stažení podepsaného dokumentu a kontrolního listu se oba dokumenty vloží opět do položkového datového modulu dokumentů k původnímu dokumentu. Typ dokumentu se určí nastavením pracovního prostoru.

4.2.2 Získání cesty k souboru

Pro získání cesty k souboru v databázi zavolám statickou metodu `OpenDocumentForView` třídy `TDMSUtils`. Tato metoda mi následně vrátí požadovanou cestu bez změny stavu dokumentu. Pokud dokument není v databázi, vrátím cestu z datového pole `FullFileCalc`.

```
function TSigniTemplate.GetDocumentFile: string;
begin
  if not ExternalDocumentDM.AsBool[DOC_IsDocumentAccesibleCalc] then
    raise K2Except.CreateChybaDM (ExternalDocumentDM, chDokFileNot);
  if ExternalDocumentDM.IsDMS then
    Result := TDMSUtils.OpenDocumentForView(
        ExternalDocumentDM.AsStringTR[DOC_FullFileCalc], False
    )
  else
    Result := ExternalDocumentDM.AsStringTR[DOC_FullFileCalc];
end;
```

Zdrojový kód 4.23: Implementace metody GetDocumentFile

4.2.3 Kontrola stavu dokumentu před odesláním

V datovém modulu dokumentů jsou implementovány tři metody pro ověřování stavu dokumentu. Jedná se o metody `CheckNotIsRevision`, `CheckNotArchived` a `CheckNotShredded`. Každá metoda ověří stav dokumentu a pokud se dokument v daném stavu nachází, vyvolá výjimku a zobrazí informativní zprávu uživateli v dialogovém okně.

```
ExternalDocumentDM.CheckNotIsRevision;
ExternalDocumentDM.CheckNotArchived;
ExternalDocumentDM.CheckNotShredded;
```

Zdrojový kód 4.24: Kontrola stavu dokumentu

4.2.4 Strávený čas

Dílčí úkol	Počet hodin
Analýza řešení	5
Implementace	15

Tabulka 4.2: Strávený čas nad úkolem Implementace do DMS

4.3 Implementace do workflow

Datový modul Workflow slouží ke snadnějšímu, přehlednějšímu a především řízenému směrování dokumentů, úkolů a informací osobám či skupinám osob (oddělením) v rámci společnosti. Poskytuje okamžitý přehled o zpracovávaných úkolech, stavu jejich řešení, odpovědnosti za jejich plnění, či prošlých termínech.

Postupy jsou v K2 obecně navržené modely firemních procesů, zachycují společné prvky těchto procesů. Obsahují sled jednotlivých kroků (úkolů). Procesy jsou již konkrétní vytvořené procesy ve firmě, které se chovají podle obecně navržených Postupů. Uživatelům se po vytvoření procesů postupně vytvářejí Kroky (jednotlivé úkoly), které mají zpracovávat [8].

4.3.1 Analýza

Pro odeslání dokumentu k podpisu skrze workflow bude zapotřebí vytvořit nový postup. V prvním kroku se bude připojovat dokument. Na vstupu druhého kroku se zavolá skript, který zobrazí formulář pro zadání scénáře podepisování. Podepisující budou ve formuláři vyplněni z řešitelů daného kroku a nebude možné je odebrat. Nebude možné ani přidávat další podepisující. Po odeslání formuláře se vytvoří kroky pro konkrétní řešitele. Jakmile dokument všichni podepíší, webhook zavolá skript, který aktualizuje stav dokumentu a odsouhlasí kroky všech řešitelů. V případě odmítnutí dokumentu se zamítnou kroky všech řešitelů a dokument se vrátí zadavateli. Dalším skriptem bude ošetřené, že řešitelé nebudou mít možnost odsouhlasit nebo zamítnout krok manuálně.

4.3.2 Vytvoření skriptu pro zobrazení formuláře

Skript má název *WkfSigniSendDocument.pas* a má jeden parametr `DocType`, ve kterém se předávají čísla typů dokumentu oddělené středníkem. K podpisu se tedy budou odesílat pouze dokumenty těchto typů, ostatní se budou ignorovat.

Ve skriptu je funkce `CurrentDM`, která vrací záznam z datového modulu, nad kterým byl skript spuštěn. Touto funkcí získám daný proces. Záznamy připojené k procesu se nacházejí v položkovém datovém modulu Produkty. Tyto produkty postupně projdu pomocí metod `SetFirst` a `DoNext`. Pokud se jedná o záznam z datového modulu dokumentů, metodou `DoGetKey` získám dokument do proměnné `LExternalDocumentDM`. Pokud je typ dokumentu v seznamu typů pro odeslání, vytvořím scénář a následně ho zobrazím ve formuláři. Takto se projdou všechny dokumenty. Po určení všech scénářů se všechny dokumenty odešlou k podpisu do Signi.

```
LProcessDM.ProductChild.SetFirst;
while LProcessDM.ProductChild.DoNext do begin
    if LProcessDM.ProductChild.DocumentDataModuleNumberId <> cD_ExternalDocument
        then continue;
    LExternalDocumentDM.Id := LProcessDM.ProductChild.DocumentNumber;
    if not LExternalDocumentDM.DoGetKey then exit;
    if not CheckTypeId then continue;
    AddSigniTemplate;
    ShowSigniTemplate;
    Inc(LIndex);
end;
SendToSigni;
```

Zdrojový kód 4.25: Procházení produktů a vytváření scénářů podepisování

Seznam řešitelů získám z položkového datového modulu Řešitelé. V `PersonId` je uloženo ID kontaktní osoby. Jako výchozí stav bude první z řešitelů navrhovatel a všichni se budou podepisovat biometrickým podpisem na konec dokumentu.

```
LProcessDM.SelectedSolver.SetFirst;
while LProcessDM.SelectedSolver.DoNext do begin
    LSigniTemplateArray[LIndex].AddSignatoryPerson(
        LProcessDM.SelectedSolver.PersonId,
        I = 0,
        TXT_Type_SigniSignatureType_Sign,
        TXT_Type_SigniSignatureLocation_End
    );
    Inc(I);
```

```
end;
```

Zdrojový kód 4.26: Procházení řešitelů a vkládání do scénáře podepisování

4.3.3 Vytvoření skriptu pro ošetření manuálního odsouhlasení nebo zamítnutí

Skript má název *WkfSigniDocumentStateCheck.pas* a má jeden parametr `aStateId`, ve kterém se předává ID stavu dokumentu k ověření.

Skript opět projde všechny produkty daného procesu. Pokud jsou všechny odeslané dokumenty ve stavu podle parametru skriptu, krok se po skončení skriptu odsouhlasí nebo zamítne. Pokud je alespoň jeden dokument v jiném stavu, nastavením globální proměnné `ExitCode` se akci zastaví a po skončení skriptu k odsouhlasení nebo zamítnutí nedojde.

```
ExitCode := ecBreak;
```

Zdrojový kód 4.27: Nastavení proměnné ExitCode

4.3.4 Vytvoření skriptu pro Signi webhook

Skript má název *WkfSigniWebhook.pas* a má oproti skriptu pro klasický webhook navíc parametry `aProcessId` pro ID procesu, `aStepRID` pro RID kroku a `aOperation` pro ID operace.

Tento skript opět nastaví dokumentu požadovaný stav. Pokud se volá webhook pro uzavření (`aOperation = 1`), skript projde všechny odeslané dokumenty a pokud jsou všechny uzavřené, odsouhlasí kroky všech řešitelů. Jestliže se volá webhook pro zamítnutí (`aOperation = 2`), tak se prvnímu řešiteli krok zamítne a tím dojde ke stornování kroků ostatních řešitelů.

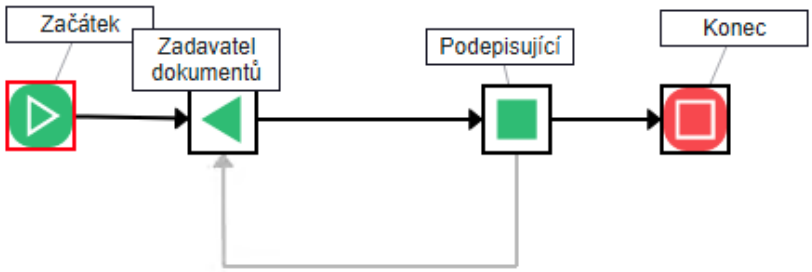
```
LProcessDM.IgnoreWorkStepSolver := True;
LProcessDM.Summary.SetFirst;
while LProcessDM.Summary.DoNext do begin
    if (LProcessDM.Summary.WorkStepRID <> aStepRID) then continue;
    if (LProcessDM.Summary.StatusId <> WSE_Created) and
       (LProcessDM.Summary.StatusId <> WSE_Read) and
       (LProcessDM.Summary.StatusId <> WSE_Accepted) then continue;
    if aOperation = 1 then
        LProcessDM.StepTerminate(
            LProcessDM.Summary.WorkStepSolverRID, WSE_Approved
        );
    if aOperation = 2 then begin
        LProcessDM.StepTerminate(
            LProcessDM.Summary.WorkStepSolverRID, WSE_Rejected
        );
```

```
        break;
    end;
end;
```

Zdrojový kód 4.28: Procházení kroků a následné odsouhlasení či zamítnutí

4.3.5 Nový postup

Vytvořil jsem nový postup s názvem Podpis Signi. Postup má čtyři kroky. První krok je Začátek, tedy vytvoření konkrétního procesu. Druhý krok s názvem Zadavatel dokumentů má vytvořenou akci typu Formulář. Do této akce jsem přidal parametr na požadovaný typ dokumentu, který se má připojit před odsouhlasením kroku. Třetí krok s názvem Podepisující má vytvořené tři akce. První je typu Vstupní. Tato akce se vyvolá při vytvoření daného kroku a spustí skript *WkfSigniSendDocument.pas*. Další dvě akce jsou typu Odsouhlasení a Zamítnutí. V obou akcích se spustí skript *WkfSigniDocumentStateCheck.pas* se zadaným parametrem, který ověří, jestli je dokument v daném stavu. Poslední krok je Konec, který ukončí vytvořený proces.



Obrázek 4.9: Nový postup – Podpis Signi

4.3.6 Úprava třídy pro podepisujícího

V metodě `InternalBeginEdit` třídy `TSignatoryPerson` vyvolám výjimku v případě, že uživatel chce ve scénáři přidávat či odebírat podepisující (týká se pouze scénářů vytvořených skriptem z procesu ve workflow).

```
if (aRecordEditMode = TRecordEditMode.remAppend) or
   (aRecordEditMode = TRecordEditMode.remDelete) then begin
    if IsFromWorkflow then
        raise K2Except.Create('Nelze přidávat ani odebírat podepisující');
```

```
end;
```

Zdrojový kód 4.29: Vyvolání výjimky při vkládání nebo odebírání podepisujících

4.3.7 Strávený čas

Dílčí úkol	Počet hodin
Analýza řešení	15
Implementace	40

Tabulka 4.3: Strávený čas nad úkolem Implementace do workflow

4.4 Funkční testy

Funkční testy slouží k rychlému odhalení chyby vzniklé především při přidávání nové funkčnosti či úpravě stávající funkčnosti. Všechny vytvořené a zaregistrované testy se spouští automaticky při každém sestavení K2. Pokud se v testu objeví chyba, zodpovědné osobě přijde informativní email s veškerými podrobnostmi. Funkční testy je možné implementovat buď v projektu K2P_Tests, který je součástí struktury K2, nebo přímo v K2 jako skriptový test.

Při vytváření nového funkčního testu se nejdříve vytvoří testovací příběh. Jedná se o dokument, který popisuje celý průběh testu. Následně se podle tohoto příběhu test implementuje.

4.4.1 Vytvoření příběhů

Všechny příběhy se ukládají do společného úložiště a vycházejí ze stejné šablony. Titulní strana dokumentu obsahuje název oblasti, do které příběh spadá, konkrétní testovanou část a verzi K2, pro kterou je příběh napsán. Za titulní stranou následuje obsah. Za obsahem se nacházejí dva odstavce. První pro popis příběhu a druhý pro přípravu K2 před spuštěním testu. Poté následují jednotlivé scénáře.

U každého scénáře je uvedený popis v několika větách. Dále jsou v bodech uvedeny jednotlivé kroky scénáře. V poslední části je v bodech uvedeno, co se má po skončení scénáře ověřit.

V rámci testování napojení K2 na Signi jsem vytvořil čtyři příběhy. Jelikož samotné podepisování dokumentu vyžaduje interakci s uživatelem, není možné testovat kompletně celý proces. Z důvodu odečítání kreditu za každý odeslaný dokument také není možné testovat odeslání do Signi. Příběhy jsou tedy následující:

- StorySigniWorkspace (testuje vytvoření nového pracovního prostoru)

- StorySigniTemplate (testuje vytvoření nové šablony pro datový modul dokumentů a pro datový modul zakázek)
- StorySigniDocument (testuje vytvoření scénáře se šablonou nad dokumentem, následné odstranění scénáře a vytvoření scénáře bez šablony)
- StorySigniSalesOrder (testuje vytvoření scénáře se šablonou nad dokumentem v příloze zakázky a následnou editaci scénáře)

4.4.2 Vytvoření testů

Samotné napojení K2 na Signi je implementováno v Delphi, proto i funkční testy budou implementovány v Delphi.

V projektu K2P_Tests jsem vytvořil složku s názvem Signi. Ve složce jsem vytvořil novou jednotku *StorySigniWorkspace.pas*. V jednotce jsem vytvořil třídu `TStorySigniWorkspace`, která je potomkem generické třídy `TStoryTest`. Tato třída určuje základní strukturu pro jeden příběh. Jako generický parametr jsem třídě předal třídu `TStoryDataWithRecords`. Tato třída slouží pro příběhy, které pracují se záznamy.

```
TStorySigniWorkspace = class(TStoryTest<TStoryDataWithRecords>)
protected
  procedure DefineStoryTest(); override;
public
  class function GetWorkspace: TSigniWorkspaceConfigAdaptable;
end;
```

Zdrojový kód 4.30: Třída TStorySigniWorkspace

Ze třídy `TStoryTest` jsem implementoval metodu `DefineStoryTest`. V této metodě jsem příběhu nastavil popis, zodpovědnou osobu za příběh a zodpovědnou osobu za testovanou oblast. Zde se také registrují jednotlivé scénáře pomocí metody `AddScenarioClass`, které se předá příslušná třída.

Statická metoda `GetWorkspace` mi vrátí instanci příběhem vytvořeného pracovního prostoru v rámci všech vytvořených prostorů.

Dle scénáře v příběhu jsem vytvořil třídu `TStorySigniWorkspace_CreateWorkspace`, která je potomkem generické třídy `TScenario`. Této třídě jsem předal opět třídu `TStoryDataWithRecords`.

```
TStorySigniWorkspace_CreateWorkspace = class(TScenario<TStoryDataWithRecords>)
const
  C_CAPTION = 'Workspace';
  C_API_KEY = 'apikey';
  C_ELECTRONIC_ADDRESS_TYPE_ID = 2;
```

```
  C_DOCUMENT_TYPE_ID = 34;
  C_SWS_URL = 'swsurl';
protected
  procedure DefineScenario(); override;
  procedure ExecuteScenario(); override;
  procedure AfterExecuteScenario(); override;
end;
```

Zdrojový kód 4.31: Třída TStorySigniWorkspace_CreateWorkspace

Ze třídy `TScenario` jsem implementoval tři metody. První z nich je `DefineScenario`, ve které se určuje popis scénáře. Druhá je `ExecuteScenario`, ve které jsou implementovány body scénáře a třetí je `AfterExecuteScenario`, ve které se ověřují správné hodnoty podle příběhu. Pro ověřování správných hodnot je ve třídě `TScenario` vytvořená property `DataChecker`. Jedná se o instanci třídy `TDataChecker`, která obsahuje několik metod pro ověřování hodnot. V tomto případě jsem použil metodu `AssertEqual`, konkrétně s parametry reálná hodnota, očekávaná hodnota a popis chyby.

```
DataChecker.AssertEqual(
    LSigniWorkspaceCA.Configuration.ApiKey, C_API_KEY, 'Nesouhlasí API klíč'
);
```

Zdrojový kód 4.32: Metoda pro ověřování hodnot

Stejným způsobem jsem vytvořil zbývající tři testy.

4.4.3 Zaregistrování testů

V jednotce *Tests.Registration.QualityAssurance.pas* jsem přidal novou konstantu pro vlákno testů napojení K2 na Signi. Následně jsem ve stejné jednotce ve funkci `AddTestThreadDefinitions` všechny testy zaregistroval.

```
LTestRegistration.Delphi.AddSimpleTestThreadRegistration(C_TEST_THREAD_SIGNI, [
    TStorySigniWorkspace,
    TStorySigniTemplate,
    TStorySigniDocument,
    TStorySigniSalesOrder
]);
```

Zdrojový kód 4.33: Metoda pro registraci testů do jednoho vlákna

Po zaregistrování se testy budou automaticky spouštět při každém sestavení K2.

4.4.4 Strávený čas

Dílčí úkol	Počet hodin
Vytvoření příběhů	10
Vytvoření testů	40

Tabulka 4.4: Strávený čas nad úkolem Funkční testy

Kapitola 5

Uplatněné teoretické a praktické znalosti a dovednosti získané v průběhu studia

Během odborné praxe jsem uplatnil hned několik teoretických i praktických znalostí, které jsem získal v průběhu studia. Uplatnil jsem znalosti z předmětů Úvod do programování (UPR), Objektově orientované programování (OOP), Databázové systémy I (DS I) a Vývoj informačních systémů (VIS).

Z předmětu UPR jsem uplatnil principy programování, které jsme probírali úplně od základu. V předmětu OOP jsem se naučil principy objektově orientovaného programování, které jsem během praxe uplatňoval po celou dobu. Z předmětu DS I jsem uplatnil znalost SQL dotazů, které jsem potřeboval především při změně struktury databázové tabulky. A z předmětu VIS jsem uplatnil znalosti ohledně obecného vývoje informačního systému a znalosti ohledně návrhových vzorů, které se v těchto systémech používají.

Kapitola 6

Scházející znalosti a dovednosti

V průběhu odborné praxe jsem se setkal především s neúplnou nebo úplně chybějící dokumentací. Když jsem narazil na určitý problém, nejdříve jsem se pokusil problém vyřešit sám a pokud se mi to do rozumné doby nepodařilo, zeptal jsem se ostatních lidí v týmu. Ti mi buď dokázali poradit, nebo mě poslali za člověkem, který se danou problematikou zabývá.

Dále jsem se ze začátku seznamoval s fungováním verzovacího systému SVN, jelikož jsem se s ním v průběhu studia nesetkal.

Kapitola 7

Dosažené výsledky a celkové zhodnocení odborné praxe

Během odborné praxe jsem dokázal realizovat veškeré požadavky zadané na začátku praxe. Implementoval jsem propojení mezi informačním systémem K2 a Signi, dále podepisování do DMS a workflow a následně jsem vytvořil několik funkčních testů. Jelikož Signi stále rozšiřuje své možnosti podepisování, bude se určitě i v budoucnu vyvíjet samotné propojení s K2.

Během praxe jsem se zapojil do vývoje jednoho z velkých informačních systémů a zjistil tak organizaci a vnitřní strukturu větších IT firem a způsob řešení a implementace požadavků zákazníků. Také jsem se naučil analyzovat a formulovat problémy a následně je řešit s kolegy a získal tak zkušenosti ohledně práce a kooperace v týmu. Protože je propojení realizováno s aplikací třetí strany, bylo také nutné komunikovat s jinou firmou během implementace, což mi přineslo také hned několik zkušeností.

Absolvování odborné praxe v K2 atmitec s.r.o. mi přineslo bohaté zkušenosti v mnoha aspektech IT. Od vývoje komplexního informačního systému, jeho propojení se systémem jiných firem a následného testování, po práci v týmu a jeho koordinaci.

Literatura

1. *Informační systém K2: podnikový software pro úspěšné firmy* [online]. Ostrava-Přívoz: K2 atmitec, c2023 [cit. 2023-03-31]. Dostupné z: `https://www.k2.cz`.
2. *Delphi (software)* [online]. San Francisco (CA): Wikimedia Foundation, 2023 [cit. 2023-04-11]. Dostupné z: `https://en.wikipedia.org/wiki/Delphi_(software)`.
3. *What is an API (application programming interface)?* [online]. Armonk: IBM, [b.r.] [cit. 2023-04-11]. Dostupné z: `https://www.ibm.com/topics/api`.
4. *What is an API?* [online]. Raleigh, North Carolina: Red Hat, 2022 [cit. 2023-04-11]. Dostupné z: `https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces`.
5. *Working with JSON* [online]. Mountain View: Mozilla Foundation, 2023 [cit. 2023-04-10]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON`.
6. *Úvod do JSON* [online]. [B.r.]. [cit. 2023-04-10]. Dostupné z: `https://www.json.org/json-cz.html`.
7. *Elektronický podpis je jen začátek* [online]. Štýřice: Signi.com, c2022 [cit. 2023-04-09]. Dostupné z: `https://signi.com`.
8. *Workflow* [online]. Ostrava-Přívoz: K2 atmitec, 2023 [cit. 2023-04-20]. Dostupné z: `https://help.k2.cz/k2iris/02/cs/134000.htm`.